AD-A092 435 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH F/G 12/1
A COMPOSITE ALGORITHM FOR MIXED INTEGER CONSTRAINED NONLINEAR O--ETC(U)
JAN 80 D B FOX
UNCLASSIFIED AFIT-CI-80-1D NL

1 OF 3
AD
A092435

UNCLASS
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER 80-1D	2. GOVT ACCESSION NO. AD A092435	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A Composite Algorithm For Mixed Integer Constrained Nonlinear Optimization.		5. TYPE OF REPORT & PERIOD COVERED ~~THESIS~~/DISSERTATION
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Capt Daniel B. Fox		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS AFIT STUDENT AT: University of Illinois		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS AFIT/NR WPAFB OH 45433		12. REPORT DATE Jan 80
		13. NUMBER OF PAGES 226
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASS
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

LEVEL II

16. DISTRIBUTION STATEMENT (of this Report)

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

DTIC
ELECTE
DEC 2 1980
S D
C

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

APPROVED FOR PUBLIC RELEASE: IAW AFR 190-17

Fredric C. Lynch
FREDRIC C. LYNCH, Major, USAF
Director of Public Affairs

Air Force Institute of Technology (ATC)
Wright-Patterson AFB, OH 45433

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

ATTACHED

80 11 24 100

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASS
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

AD A092435

DDC FILE COPY

A COMPOSITE ALGORITHM FOR MIXED INTEGER CONSTRAINED NONLINEAR OPTIMIZATION

Daniel B. Fox
CAPT. USAF

Department of Mechanical and Industrial Engineering
University of Illinois at Urbana-Champaign, 1980

Pages: 226 Degree awarded: PhD

A composite optimization algorithm applicable to mixed integer, constrained, nonlinear problems is developed in this research. One major component of the composite algorithm is a modified version of the nonlinear simplex method. Significant modifications are made to this algorithm including the incorporation of a unidimensional search procedure and the use of a new method to treat constraints. Additional features of the composite algorithm include new acceleration strategies, a new decomposition approach, and a discrete grid algorithm.

The components of the composite algorithm are tested on problems primarily selected to represent engineering design optimization applications. The performance of the new methods is compared to some existing techniques. Examples of the application of combinations of the composite components are included. The results indicate that the new algorithms obtain superior solutions and in most cases are more efficient than existing techniques. The success of the algorithm on problems of engineering design optimization indicates a wide area of potential application.

-B

A COMPOSITE ALGORITHM FOR MIXED INTEGER CONSTRAINED NONLINEAR OPTIMIZATION

BY

DANIEL B. FOX

B.S., University of Illinois, 1969
M.S., Oklahoma State University, 1970

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Mechanical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1980

Urbana, Illinois

A COMPOSITE ALGORITHM FOR MIXED INTEGER CONSTRAINED NONLINEAR OPTIMIZATION

Daniel B. Fox
CAPT. USAF

Department of Mechanical and Industrial Engineering

University of Illinois at Urbana-Champaign, 1980

Pages: 226 Degree awarded: PhD

A composite optimization algorithm applicable to mixed integer, constrained, nonlinear problems is developed in this research. One major component of the composite algorithm is a modified version of the nonlinear simplex method. Significant modifications are made to this algorithm including the incorporation of a unidimensional search procedure and the use of a new method to treat constraints. Additional features of the composite algorithm include new acceleration strategies, a new decomposition approach, and a discrete grid algorithm.

The components of the composite algorithm are tested on problems primarily selected to represent engineering design optimization applications. The performance of the new methods is compared to some existing techniques. Examples of the application of combinations of the composite components are included. The results indicate that the new algorithms obtain superior solutions and in most cases are more efficient than existing techniques. The success of the algorithm on problems of engineering design optimization indicates a wide area of potential application.

80-1

AFIT RESEARCH ASSESSMENT

The purpose of this questionnaire is to ascertain the value and/or contribution of research accomplished by students or faculty of the Air Force Institute of Technology (ATC). It would be greatly appreciated if you would complete the following questionnaire and return it to:

AFIT/NR
Wright-Patterson AFB OH 45433

Research Title: A Composite Algorithn for Mixed Integer Constrained Nonlinear Optimization

Author: Capt Daniel B. Fox

Research Assessment Questions:

1. Did this research contribute to a current Air Force project?

a. Yes b. No

2. Do you believe this research topic is significant enough that it would have been researched (or contracted) by your organization or another agency if AFIT had not?

a. Yes b. No

3. The benefits of AFIT research can often be expressed by the equivalent value that your agency achieved/received by virtue of AFIT performing the research. Can you estimate what this research would have cost if it had been accomplished under contract or if it had been done in-house in terms of manpower and/or dollars?

a. Man-years ______. b. $______

4. Often it is not possible to attach equivalent dollar values to research, although the results of the research may, in fact, be important. Whether or not you were able to establish an equivalent value for this research (3 above), what is your estimate of its significance?

2. Highly Significant b. Significant c. Slightly Significant d. Of No Significance

5. AFIT welcomes any further comments you may have on the above questions, or any additional details concerning the current application, future potential, or other value of this research. Please use the back of this questionnaire for your statement(s).

NAME GRADE POSITION

ORGANIZATION LOCATION

USAF SCN 75-20B

ACKNOWLEDGEMENTS

The author expresses his sincere appreciation to his advisor, Professor Judith S. Liebman for advice and encouragement provided throughout this research. Suggestions from Professors N. Kachaturian, N. Miller, and C. Pedersen proved exceptionally valuable and are also appreciated. Special thanks are due my wife for her patience and assistance. Financial support of the United States Air Force and the University of Illinois Research Board is acknowledged.

TABLE OF CONTENTS

1 INTRODUCTION

1.1 Purpose

Many engineering design problems can be represented as constrained nonlinear programming problems where all or some variables are restricted to discrete values. The purpose of this research is to develop a practical method for solving mixed integer constrained nonlinear optimization problems and to demonstrate the technique on some problems of machine, thermal systems, and civil engineering design. The basis for the optimization technique is the nonlinear simplex method.

One important factor to consider when deciding if an optimization method is practical is whether practitioners in need of an optimization technique are likely to use the algorithm. This decision is often a function of whether the practitioner, not necessarly a specialist in optimization, can understand the algorithm. Nonlinear programming is not reliable in the way that linear programming is reliable. In linear programming a properly formulated model will yield an optimal solution using any of a number of available linear programming packages. In nonlinear programming, except for a few classes of well behaved problems, finding the optimal solution depends on interaction with the user in order to avoid some solutions that may be only local optima or not optimal at all. This necessitates some familiarity by the user with the internal workings of the algorithm as we'l as with the model being optimized. For this reason, a simple algorithm is often more desirable than a more complicated one, even at the cost of some efficiency.

In this research a heuristic algorithm is developed for the solution of mixed integer, constrained, nonlinear programming problems. Straightforward methods have been used to enforce the constraints and discreteness of variables. A variety of search strategies, each employing easily understood concepts, is combined into a methodology which allows the designer to change strategies as the algorithm proceeds. The whole of the options and strategies available make up a composite algorithm.

The elements of the composite algorithm are tested using both classical test functions as well as engineering design problems selected from recent literature. Results are compared with results from existing techniques.

1.2 Statement of the Problem

The general integer, constrained, nonlinear optimization problem is usually stated as:

MINIMIZE: $F(X)$

SUBJECT TO: $Gi(X) \leq 0$, $i = 1,\ldots,m$

$Hi(X) = 0$, $i = 1,\ldots,k$

$X(j)$ integer , $j = 1,\ldots,n$

The problems considered in this research may be stated as:

MINIMIZE: $F(X)$

SUBJECT TO: $Gi(X) \leq 0$, $i = 1,\ldots,m$

with a specified subset of X discrete

In this research equality constraints are excluded for two reasons. First, the complex algorithm that is a major component of the composite algorithm developed in this research does not allow equality constraints. Second, as discussed by Taha [72], the inclusion of equality constraints in discrete problems can bring into question the very existence of solutions to what would be problems with many solutions if either the equality constraints, or the discrete requirement were dropped. However, the techniques developed in this research do allow the solution of problems with equality constraints in certain circumstances. There is always the possibility of solving an equality constraint for one of the variables of the problem. Then the optimization problem can be restated, without the equality constraint, in a reduced variable space. This procedure includes what is a very common situation in structural optimization where the objective function is written in terms of the design variables but some of the constraints may be written in terms of state variables. The design variables are linked to the state variables by equality constraints. In this case, since the state variables do not enter into the calculation of the objective function, the equality constraints may be considered to be a transformation of the design variables into the state variables. This transformation may be in closed form or accomplished by simulation. Thus, the constraints written in terms of the state variables may be considered to be functions of the design variables after the transformations have been applied. The optimization problem can then be expressed without equality constraints.

In this research it is possible for only a subset of the variables to be integer restricted while the remaining variables are allowed to assume continuous values; and it is possible for discrete variables to assume any set of equally spaced values, not necessarily integer. Of course, a discrete variable that assumes nonequally spaced values can always be transformed to one that is equally spaced. The subset of variables restricted to discrete values may be the entire set of variables of the problem, resulting in an all discrete problem, or may be the empty set, resulting in an all continuous problem.

Although the objective function F and the constraint functions Gi are often analytic expressions, the algorithm developed in this research requires only that they be computable. Hence, these function values may result from recursive calculations, simulation, or represent the output of a so called "black box," a system which provides outputs when given inputs, but for which the internal workings are unknown.

1.3 Notations and Conventions

Certain conventions of the FORTRAN computer language will be used in representing mathematical operations. For example:

*	is used to denote multiplication
/	is used to denote division

The angle brackets notation <a> is used to denote the nearest integer value to a.

Examples: <1.3> = 1   <.2> = 0   <-1.3> = -1   <-2.8> = -3

For scaler a and vector V, where:

W(i) = a * V(i) for all i

the vector W is denoted

W = a * V

or

$$W = V * a$$

All of the optimization problems discussed in this research are stated as minimization problems. Thus, the optimal solution sought is the point with the smallest value of the objective function that satisfies the constraints of the problem. For clarity in exposition, the phrase "point with smallest value of the objective function" is sometimes replaced with "best point." Similarly, the phrase "worst vertex" should be interpreted as "the vertex with the largest value of the objective function."

1.4 Definitions

This section provides definitions of terms used in the discussion to follow.

Definition: Parameter space

The parameter space for an optimization problem with n variables is a vector space of real n-tuples.

Definition: Subspace

A subspace is defined as the domain of a subset of the variables.

Definition: Discrete subspace

The discrete subspace is the subspace of the discrete variables.

<u>Definition</u>: Continuous subspace

The continuous subspace is the subspace of the continuous variables.

<u>Definition</u>: Increment

An increment is the distance between discrete points along a coordinate axis.

The primary reason for the definition of increments for the discrete variables is to define the lattice of values that the discrete varibles may assume. As will be seen later, the increment is also used in calculating termination criteria for both the modified complex algorithm and the unidimensional search. In addition, the increment specifies the stepsize used to calculate gradient approximations. These last three uses of the increment prompt the following definition of a pseudo-increment for continuous variables.

<u>Definition</u>: Pseudo-increment

A pseudo-increment is the smallest distance along a coordinate axis of a continuous variable that is significant. The determination of significance is the responsibility of the designer.

The pseudo-increment serves the same functions as the increment with the exception of defining a lattice. That is, the pseudo-increment is used to calculate termination criteria and gradient approximations but does not restrict the associated variables to a lattice.

<u>Definition</u>: Grid point

A grid point is a node of the lattice of the discrete subspace. The values of the continuous variables are fixed, but arbitrary.

<u>Definition</u>: Discrete unit neighborhood

Let I(i) be the increment of the (i)th variable. Let X be a grid point with discrete components X(i) and continuous components X(j).

The discrete unit neighborhood of a point X, UN(X), is a set of points Y such that:

Y is a grid point with discrete components Y(i) and continuous components Y(j)

and

$X(i) - I(i) \leq Y(i) \leq X(i) + I(i)$ for all discrete variables i

and

$X(j) = Y(j)$ for all continuous variables j

<u>Definition</u>: Discrete diagonal neighborhood

The discrete diagonal neighborhood of a point X, DN(X), is a set of points Y such that:

Y is a member of the discrete unit neighborhood of X

and

$X(i) \neq Y(i)$ for any discrete variables i

<u>Definition</u>: N1 discrete neighborhood

The N1 discrete neighborhood of a point X, N1(X), is the set difference between the discrete unit neighborhood of X and the discrete diagonal neighborhood of X. That is:

{N1(X)} = {UN(X)} - {DN(X)}

Example: Neighborhoods

The neighborhoods defined above are illustrated in two dimensions ir Figure 1-1. In this figure:

{UN(X)} = {A,B,C,D,E,F,G,H,X}

{DN(X)} = {A,C,F,H}

{N1(X)} = {B,D,E,G,X}

<u>Definition</u>: Explicit constraints

Explicit constraints are constraints functions that may be written in terms of a single variable. These are constraints that express upper or lower bounds on variables.

<u>Definition</u>: Implicit constraints

Implicit constraints are constraint functions that are functions of two or more variables.

<u>Definition</u>: Feasible point

A feasible point is a grid point for which all the explicit and implicit constraint functions are satisfied.

<u>Definition</u>: Effective objective function

The effective objective function EF(X) is the modified function minimized by the composite algorithm developed in this research. It is developed in three stages:

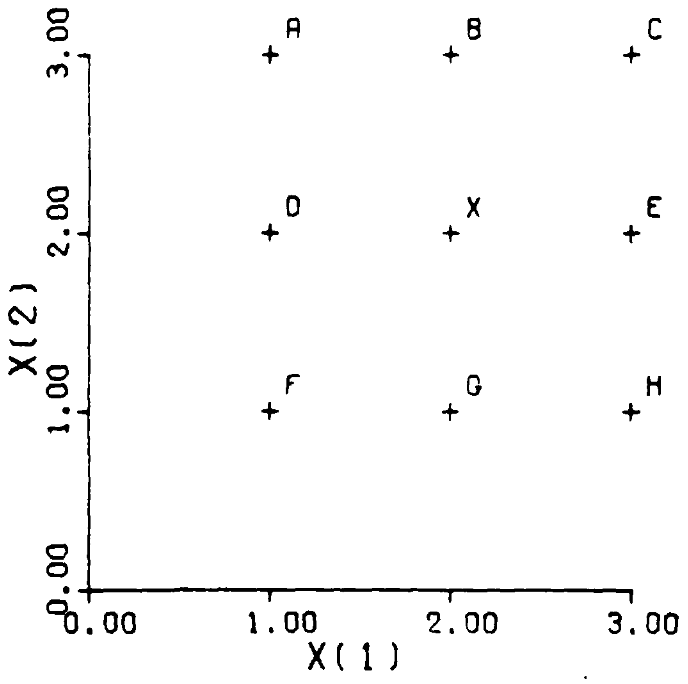


FIGURE 1-1. NEIGHBORHOODS

STAGE 1

If any explicit constraints are violated at X then:

EF(X) = M2 + (violations)

where

M2 is a large number

and

(violations) is the sum of the violations of the explicit constraints at X.

STAGE 2

Assuming the explicit constraints are satisfied, then:

If any implicit constraints are violated at X let:

EF(X) = M1 + (violations)

where

M1 is a large number

and

(violations) is the sum of the violations of the implicit constraints at X.

STAGE 3

Assuming both the explicit and implicit constraints are satisfied then

EF(X) = F(X)

where

F(X) is the original objective function of the problem.

<u>Definition</u>: Vicinity

The definition of vicinity depends on whether discrete variables are present. If the variables are all discrete, then the vicinity is the N1 neighborhood. If all variables are continuous, then the vicinity is a small hypersphere. In the case of mixed discrete and continuous variables, the vicinity is the union of the vicinity of the discrete subspace and the vicinity of the continuous subspace.

<u>Definition</u>: Local optimum

A feasible point X is a local optimum if:

$F(X) \leq F(Y)$ for all feasible Y in the vicinity of X

<u>Definition</u>: Global optimum

A feasible point X is a global optimum if:

$F(X) \leq F(Y)$ for all feasible Y.

<u>Definition</u>: Complex

A complex is a figure in n-space represented by n+1 (or more) points.

<u>Definition</u>: Regular simplex

A regular simplex in n-space is a complex of n+1 points with the distance between a pair of points equal to the distance between every other pair of points.

<u>Definition</u>: Centroid

The centroid of a set of points X1,X2,...,Xn is a point C such that:

$$C(i) = (X1(i) + X2(i) + \ldots + Xn(i)) / n$$

Thus each coordinate of the centroid is the arithmetic mean of the corresponding coordinate of the set of points defining the centroid.

1.5 Guide to the Remaining Chapters

Chapter 2 introduces the topic of design optimization and discusses the treatments that have been used for constraints and discrete variables. The nonlinear simplex algorithm and its major modifications are reviewed.

Chapter 3 describes the elements of the composite algorithm developed in this research. The new modifications for the complex algorithm, the auxiliary techniques, and additional search strategies are described in detail. The chapter concludes with a detailed description of the modified complex algorithm.

Chapter 4 contains an analysis of the results and is divided into six sections. After the parameter settings used to obtain the results are described in Section 1, the results of the modified complex algorithm on four categories of problems are analyzed. The four categories are: constrained discrete, constrained continuous, unconstrained discrete and unconstrained continuous. The last section discusses results of the additional features of the composite algorithm.

Chapter 5 summarizes the results and suggests areas for additional research.

2 REVIEW OF THE LITERATURE

2.1 Introduction

Design has historically been a trial and error process based on experience with similar designs. Gradually techniques for analysis have been devised which have eventually become accurate enough to predict performance of designs based on descriptions of the design. As the analysis techniques have become more comprehensive, the computer has been used to perform the computations necessary for the analysis. An additional step towards automation was taken when improved means were provided for the interaction of the designer with the computer. This "computer aided design" allows the designer to indicate design changes which the computer analyzes and for which the results are displayed. Thus, the designer can evaluate the results of the analysis, make design changes, and iteratively arrive at a design that is in some sense optimal. Recently there has been increased interest in closing the design loop within the computer by including in the analysis a measure of merit or objective function which can be used to compare various designs. A computer program can then vary the design parameters and seek the parameters that define the optimal design as measured by the objective function. Of course, an objective function represents only an approximate measure of the utility of a design, and in actual fact, the selection of the final design must be tempered by the experience of the designer. The worth of the optimization process lies in its ability to optimize within limits set by the designer and in providing alternative designs.

In this final phase, the design problem is cast in the form of a mathematical optimization problem. Virtually every mathematical optimization technique has been applied, including purely analytical techniques, linear programming, dynamic programming, and direct search. Depending on the nature of the problem, auxiliary techniques including Lagrange multipliers, penalty functions, linearization, and rounding have all been used to aid in the solution of problems.

2.2 Design Optimization

Some progress has been made in analytical optimization in structural design, for example, in fully stressed design [8] and more recently in the interpretation of failure modes of structures as representing local optimal solutions of an optimization problem [36]. These approaches, however, require intensive analysis of the particular problem under study and cannot be considered to be generally applicable methods. Instead, the approach in this research is limited to the use of a general purpose optimization algorithm on a model of the design to be optimized.

Good introductions for those not familiar with how a mathematical optimization problem is formulated from design concepts are in Schmit [62], Fox [21] and Gallagher [23]. Two survey articles on the current state of the art in optimal structural design, which include very extensive references, are by Wasiutynski and Brandt [75] and Sheu and Prager [65].

The success of many nonlinear optimization techniques is predicated on the problem being in a certain form, whereas most design problems result in a model of unknown form. Conditions of continuity, differentiability, convexity or freedom from numerical difficulties in the computation of the objective function or constraints most often cannot be assured in typical engineering design applications. A class of nonlinear optimization techniques that makes few assumptions about the form of the problem is the class of direct search techniques. Most optimization algorithms, particularly those that attain efficiency by extensive use of the local topography of the objective function surface to determine search directions, will converge to a local optimum nearest the starting point [5]. This, of course, includes the gradient based optimization methods. For this reason, when the model to be optimized contains multiple local optima, use of a direct search optimization method may be preferable. Direct search has been demonstrated to be appropriate for design optimization by Pappas and Allentuch [48], Pappas and Amba-Roa [49], de Silva [14], and Weisman and Wood [76]. A particular direct search algorithm, the simplex method, has been cited for having the potential for locating optimal solutions far from the starting point [5].

2.2.1 Constraints in Design Optimization

Constraints in engineering design optimization have been widely treated with penalty functions [57] [56] [44] [40]. Interestingly, an early paper developing the mathematical basis for the penalty function concept was written in 1943, well before the use of computer optimization techniques [12]. The mathematics are developed by considering the equilibrium for a plate or membrane under an external force. The technique is based on replacing an unsolvable or difficult problem P with an easily solved problem P1 with solution S1. The approximation is then improved, resulting in problem P2 with solution S2. Again, the approximation is improved and the result is a series of problems Pn with solutions Sn. If the sequence of problems is appropriately selected, two useful properties result. First, knowledge of the solution to the (n)th problem aids in the solution of the (n+1)st problem. Second, the sequence of solutions Sn tends to the solution of the problem P.

The use of penalty functions transforms a constrained nonlinear optimization problem into a sequence of unconstrained nonlinear problems. This is accomplished by augmenting the objective function of the constrained problem with penalty terms. The penalty terms are formed from the constraints and multiplied by a penalty function parameter. The augmented function is then optimized, using unconstrained optimization techniques, for a sequence of values of the penalty function parameter. The form of the penalty terms and the sequence of the penalty function parameter values are chosen such that the sequence of solutions of the unconstrained problems converges to an optimal solution of the original constrained problem [1] [32]. Penalty function methods may be classified into two catagories usually called internal and external. Some algorithms employ a mixture of the two.

Exterior penalty function methods are sometimes referred to as dual methods because they locate a series of optimal, infeasible solutions and terminate when a feasible solution is found [16] [32] [78]. The use of "dual" as a name alludes to the similarity in approach to the dual method of linear programming. The exterior penalty method (and the mixed methods as well) suffer a defect common to dual methods in that if the algorithm is terminated prior to completion, for example due to budget limitations on computer usage or due to numerical difficulties in the calculations, then no feasible solution may be available. Another difficulty with exterior penalty function formulations is that it is necessary to evaluate the objective function for infeasible points. In some cases this may be impossible or may lead to numerical difficulties in computation.

Interior penalty function methods, on the other hand, are often referred to as primal methods because they begin with a feasible solution and attempt to locate an optimal solution while retaining feasibility [8] [16] [32]. Interior penalty methods have the advantage of always having feasible, and hence potentially usable, solutions as intermediate results. Discussions of the use of penalty function methods specifically applied to design optimization have been written by Moe [44] and also by Fox [21].

In spite of wide usage, penalty functions methods have some potentially serious difficulties associated with them. As Schuldt et. al. [64] observe, the problems of scaling the objective function and constraints, selecting an appropriate initial penalty factor, and determining the amount to reduce the penalty factor between cycles, are problems for which general procedures have not been developed. Improper scaling between multiple constraints or between the objective function and the constraints can result in the penalty function method converging to a nonoptimal solution. A more basic problem is that the addition of the penalty term to the objective function may mean that the assumptions implicit in the unconstrained optimization algorithm are violated. In particular, Lasdon, Fox and Ratner [39] comment that unidimensional search techniques which are commonly used as part of most unconstrained optimization algorithms and which are based on polynomial approximation are probably inappropriate for objective functions that include penalty terms. Davies and Swann [13] observe that penalty functions insure the presence of steep valleys and discontinuous derivatives, and that these features are difficult to overcome, particularly with gradient based algorithms. Murray [45] notes that the use of penalty functions makes the problem progressively more ill-conditioned as the solution is approached, and that convergence is strongly dependent on the selection of the penalty parameters.

Finally, since the penalty function method requires solving a sequence of problems with different values of the penalty parameter, the overall process may be inefficient.

2.2.2 Discreteness in Design Optimization

Until recently, discreteness in design optimization has been most often treated either explicitly by branch and bound or dynamic programming or implicitly by rounding to a nearby integer solution. More recently, some explicit treatments of discreteness in search techniques have been attempted [10] [24] [25] [26] [40] [48] [66].

The branch and bound algorithm was originally proposed by Land and Doig [38] for linear integer problems. The basic concept is to enumerate the integer solutions in such a way that groups of solutions, which cannot contain the optimal solution, may be recognized without detailed evaluation of all combinations of the discrete variables. The nonoptimal groups are identified by solving a linear subproblem and calculating a limiting value for the solutions in that group (bounding). When the limiting value is no better than a known solution then the group cannot contain the optimal solution. By recognizing these nonoptimal groups and dismissing those solutions from further consideration, an optimal solution may be found after evaluating only a fraction of the total number of solutions represented by all combinations of the discrete variables. Branch and bound algorithms have been successful on linear integer and mixed integer problems because efficient and reliable algorithms exists with which to solve the linear subproblems.

In theory, the concept of the branch and bound technique may equally well be applied to nonlinear integer problems. Unfortunately, except for problems where stringent conditions on the form of the objective function and constraints are known to hold, the situation for nonlinear problems is quite different from that for linear integer problems. The subproblems are now nonlinear, constrained, optimization problems and solving them may be relatively difficult. More importantly, the solution obtained may be only a local rather than the global optimum. Even a small integer problem may involve solving 100 subproblems. Any one local optimum for a subproblem, which is wrongly presumed to be a global optimum, may result in erroneously discarding a group of solutions as not containing the optimum when, in fact, the optimum may be in that group.

Thus, using a nonlinear programming algorithm to solve the branch and bound subproblems and failing to locate the global solution even occasionally may result in the branch and bound approach finding suboptimal solutions to the overall problem. In addition, the sheer number of nonlinear subproblems to be solved for even small problems can make the branch and bound approach infeasible.

Dynamic programming is limited to problem formulations that are separable and becomes computationally inefficient when the number of constraints exceeds four or five. Since separability is not generally present in models for engineering optimization, dynamic programming is not a generally applicable optimization technique. For this reason it is not considered further in this research.

The simplest and most prevalent technique for treating integer variables is some form of rounding of the continuous optimum. This is often combined with a search of the neighborhood of the rounded solution [21] [31] [35] [41] [73] [48]. Of course, obtaining a continuous optimum is possible only if the discrete variables of the problem can satisfactorily be treated as continuous. For example, if the value of a variable denotes the use of a certain material (with specific properties that affect the objective function and constraints) then it is not clear what is implied by the variable taking on a noninteger value. That the rounded solution, even with neighborhood search, may not be optimal is widely recognized [4] [27] [48] [21] [24]. A major difficulty in constrained problems is that the rounded solution may not only be suboptimal but may actually be infeasible. Locating a feasible discrete point may in itself be a nontrivial task.

Treating the discreteness of the variables explicitly in a search algorithm has the advantage of only requiring evaluation of the objective function and constraints on the allowable set of discrete points. It is this method that is used in this research and the details of the method are presented in the following chapter.

2.3 The Nonlinear Simplex Method

The nonlinear simplex method, not to be confused with the simplex algorithm of linear programming, is a direct search, descent method. It is based on a geometric construct referred to, in the case of an n-dimensional space, as an n-dimensional simplex. The original work published in 1962 by Spendley, Hext, and Himsworth [70] was based on regular simplices, that is simplices with all line segments of the same length. Although the particular application was to a response surface problem, the authors alluded to the applicability of the technique to mathematical optimization.

The simplex method for minimization may be summarized as follows:

1. Construct a regular simplex in the parameter space of the variables of optimization and evaluate the objective function at each vertex.

2. Find the centroid of the simplex without the worst vertex (worst vertex being the vertex with the highest objective function value).

3. Define a new simplex by eliminating the worst vertex and adding a new vertex obtained by reflecting the worst vertex through the centroid. Evaluate the objective function at the new vertex and return to step 2.

The elegant simplicity of this algorithm has led to a large number of variations on the basic method.

2.4 Simplex Variations

Variations in the simplex method have been made to adapt the technique as a general purpose mathematical optimization technique. Additional modifications have been proposed to allow for optimizing constrained problems and still other modifications to adapt the algorithm to discrete problems. Various minor modifications have been proposed to improve the performance of the algorithm. These modifications are discussed below.

2.4.1 Nelder and Mead Algorithm

In 1964 Nelder and Mead [46] extended the simplex technique by allowing irregular simplices (line segments not necessarly all the same length). This provided both a degree of scale invariance and allowed for a form of acceleration in the search. They also advocated using more than n+1 points as an aid in preventing the simplex from collapsing into a subspace. Their modifications to the simplex rules include:

1. If the reflected vertex is the best vertex (best vertex being the vertex with the lowest objective function value), then try an expansion step to another vertex that is further along in the same direction that yielded the reflected vertex.

2. If the reflected vertex has the worst objective function value, then try another vertex that is retracted toward the centroid.

3. If the retracted vertex has the worst objective function value, then contract the entire simplex by moving every vertex towards the best vertex.

4. Stop the process when the simplex shrinks to a sufficiently small size.

2.4.2 Box's Complex Algorithm

In 1965 Box [5] proposed a modified simplex method called the complex method that could solve constrained problems with an interior, that is, problems constrained only by inequalities. In this algorithm the vertexes of the simplex are constrained to remain within the feasible region by the following rules:

1. If a reflected vertex violates an explicit constraint (a variable does not fall within its lower and upper bounds), then that variable is set just within its violated bound.

2. If the above rule is satisfied but an implicit constraint (a constraint other than a simple bound on a variable) is violated, then the vertex is retracted towards the centroid until all constraints are satisfied.

2.4.3 A Discrete Complex Algorithm

Beveridge and Schechter [4] suggest modifications of the complex method of Box to solve integer nonlinear problems. This method was further modified by Glankwahmdee [25] and

Glankwahmdee, Liebman and Hogg [26]. In Glankwahmdee's algorithm the following additional rules are used:

1. Each vertex is restricted to be at an integer point by moving to the nearest discrete point.

2. A reflected point is retracted towards the centroid if it either violates a constraint or has the worst objective function value.

3. If a point in being retracted coincides with either the original reflected point or the original vertex to be rejected, then the original vertex is restored and the centroid is located and a reflected point is determined using the second worst vertex (third worst vertex, etc).

4. The process is terminated either when the simplex contracts to a single point or when rule 3 uses all vertexes in the simplex without a vertex being changed.

2.4.4 Unlimited Expansion Modification

Parkinson and Hutchinson [52] report a successful variant of the Nelder and Mead algorithm that allows for repeating the expansion step as long as the expansion vertex is the best vertex.

2.4.5 Modifications to the Algorithm

Guin [29], after some experiments with Box's algorithm suggested that the rule for setting variables just inside their bounds sometimes caused premature termination of the algorithm. For this reason it was suggested that this rule be abandoned. Instead, explicit constraint violations were treated just as implicit constraint violations and the rule calling for retraction towards the centroid was applied. Also suggested was that, for cases where the centroid is infeasible (which can occur with non-convex constraint sets), the entire complex should be contracted towards the best vertex.

2.4.6 A Nonrandom Initialization Procedure

The various simplex based algorithms discussed here usually suggest a random procedure of some sort to define the initial simplex. In contrast, Mitchell and Kaplan [43] suggest a nonrandom method for defining the initial simplex. The nonrandom simplex of 2n+1 points is defined, given an initial feasible point X0, as follows:

$$V(1,i) = X0(i) \;,\; i = 1,\ldots,n$$

$$V(k+1,i) = X0(i) \;,\; i = 1,\ldots,n \;,\; i \neq k \;,\; k = 1,\ldots,n$$

$$V(k+1,k) = LB(k) \;,\; k = 1,\ldots,n$$

$$V(1+n+k,i) = X0(i) \;,\; i = 1,\ldots,n \;,\; i \neq k \;,\; k = 1,\ldots,n$$

$$V(1+n+k,k) = UB(k) \;,\; k = 1,\ldots,n$$

where:

LB(k) is the lower bound for variable k

and:

UB(k) is the upper bound for variable k

If any of the vertexes defined above are infeasible they are retracted half way toward the initial point as many times as is necessary to locate a feasible point.

2.5 A Discrete Search Algorithm

2.5.1 Unconstrained Discrete Search

Glankwahmdee [25] and Glankwahmdee, Liebman and Hogg [26] reported that an algorithm of descent along the integer gradient combined with sectioning regeneration was the best of several algorithms developed for unconstrained discrete optimization. This algorithm, G-2, may be described as follows:

1. Calculate a gradient approximation at the current solution point.

2. Convert the gradient direction to an integer direction. This procedure is described in Chapter 3, under "A New Method for Search Direction Specification."

3. Perform a unidimensional search along the integer direction.

4. If the search locates a better point, make it the current solution and go to step 1; otherwise, go to step 5.

5. Perform a regeneration step by applying the unidimensional search along each of the coordinate directions in turn until a better point is found or all coordinate directions are tried.

6. If a better point is found, make it the current solution and go to step 1; otherwise, go to step 7.

7. Stop.

2.5.2 Constrained Discrete Search

Chanaratna [10] and Chanaratna, Liebman, and Khachaturian [40] report successes in the optimization of some structural designs by adding interior penalty functions to Glankwahmdee's G-2 algorithm. This algorithm, G-2/P, uses a variant of the normal interior penalty function formulation to allow the solution of discrete problems where a constraint may be satisfied exactly at the optimal solution. This is done by replacing the penalty term for a particular constraint G(X):

$$PF = 1./G(X)$$

by

PF = 1./G(X)   when G(X) $\neq$ 0.

or

PF = 1./EPS   when G(X) = 0.

where EPS is a small constant. This modification prevents the penalty term from being infinitely large when G(X) equals zero.

3 DESCRIPTION OF THE ALGORITHM

3.1 Introduction

The goal of this research is to develop and test a practical algorithm for design optimization where the mathematical representation of the system to be optimized may have a linear or nonlinear objective function, linear or nonlinear constraints and some or all variables restricted to discrete values. No assumptions are made concerning the continuity, convexity or differentiability of the objective function or constraints. The general nature of this problem and the inclusion of discrete variables make obtaining the optimal solution of the problem difficult. The approach adopted to solve this problem was suggested by Taha [72].

> "Experience with practical integer problems shows that if a solution is to be found, manual intervention during the course of the computations is a must. This means that, depending on the progress of the calculations, the user may find it necessary to change search strategy in order to take advantage of the available information. This emphasizes the importance of including as many feasible options as possible in the integer

> programming code. These options should be designed to exploit the different techniques available for solving integer programming problems, including heuristics. The collection of these options, together with manual intervention by the user, produce the so-called "composite" algorithm. Naturally, the specific steps of the algorithm are not fixed in advance but will primarily depend on the experience of the user in selecting the most effective strategies for directing the search toward finding the optimal solution. These strategies are usually based on the information feedback from the computer and also on the type of problem under investigation."

The major component of the composite algorithm developed in this research is the nonlinear complex method incorporating both new modifications and modifications to the basic algorithm previously reported in the literature but not previously combined.

The complex algorithm, five new modifications, four auxiliary techniques, two additional search algorithms, and the ability to change at will the algorithm parameters, are implemented as an interactive, terminal-oriented, composite computer program. The new modifications include the following:

1. Incorporation of a unidimensional search procedure into the complex algorithm.

2. A new method for specifying discrete points along a search direction.

3. A new method for handling constraints.

4. A new termination criterion for the unidimensional search component.

5. A new termination criterion for the complex algorithm.

The four auxiliary techniques for use in conjunction with the modified complex algorithm are:

1. Regeneration methods

2. Acceleration strategies.

3. A new decomposition method.

4. A grid approach.

The two additional search algorithms included in the composite algorithm are:

1. Steepest descent

2. Sectioning

What follows is a description of the modifications of the complex algorithm, the auxiliary techniques, and the additional search algorithms. The chapter is concluded with a detailed description of the modified complex algorithm.

3.2 Additional Modifications

3.2.1 Incorporation of Unidimensional Search

The reflection, expansion, and contraction rules, for all the variations of the complex algorithm discussed in the previous chapter, select new vertexes that are linear combinations of a selected vertex of the current complex and the centroid of the remaining vertexes. Thus, new vertexes always lie on a line connecting the centroid and the selected vertex. In this research these three rules have been replaced by a unidimensional search. In general, the base point of the search is taken as the selected (usually the worst) vertex and the direction is defined as the line from the selected vertex to the centroid of the remaining vertexes. This formulation allows for movements similar to reflection, expansion and retraction, in addition to multiple expansions, as allowed in the Parkinson and Hutchinson [52] algorithm, and for multiple retractions as specified for infeasible points in Box's [5] algorithm.

The unidimensional search used is the golden section method. The search method is comprised of two steps: determining an interval within which the search will be made (this is refered to as bracketing) and the search itself.

The bracketing procedure selects trial points at geometrically increasing distances along the specified direction and is described in detail by Avriel [1]. The goal of the bracketing procedure is to obtain three points that satisfy the following inequality:

$$F(X1) > F(X2) < F(X3)$$

Under conditions of unimodality of the objective function F, the minimum of the objective function is known to be between X1 and X3 if this inequality is satisfied. In general, although unimodality is not known, a pretense of unimodality is assumed. As a result of this assumption a local, rather than global, minimum may be bracketed. The bracketing procedure may be summarized as follows:

```
Let:    X0 be a given base point
        D  be a direction vector
        S  be the initial stepsize
        X1 = X0
        X2 = X1 + S * D
```

One of two cases are possible at this point. If $F(X2) < F(X1)$ then the objective function is decreasing in the direction D and one may proceed.

If the above inequality does not hold, then the objective function is not decreasing in the direction D. In this case, the direction of search is reversed by the following:

Interchange X1 and X2

Let:

$S = -S$

In any case repeat the following steps as required.

Let:

$S = 2. * S$

$X3 = X2 + S * D$

If $F(X3) > F(X2)$ stop because then the bracket is accomplished. Otherwise let:

$X1 = X2$

$X2 = X3$

The search procedure attempts to decrease the bracket size so as to localize the minimum of the function along the specified direction. Again, at least the pretense of unimodality must be assumed. The golden section search method and its relation to the Fibbonacci search method is extensively discussed by Wilde and Beightler [77]. The procedure may be summarized as follows:

Suppose the current bracket is at X1, X2, and X3 where:

$F(X1) > F(X2) < F(X3)$

Without loss of generality assume:

$X1 < X2 < X3$

Case 1:

X2 is in the left half of the interval between X1 and X3

Let:

```
     XNEW(i) = X1(i) + .618 * (X3(i) - X1(i)) , i = 1,...,n
```

Define a new bracket as follows:

```
     If F(XNEW) < F(2)
     then
          X1 = X2
          X2 = XNEW
     else:
          X3 = XNEW
```

Case 2.

X2 is in the right half of the interval between X1 and X3

Let:

```
XNEW(i) = X3(i` + .618 * (X1(i) - X3(i)) , i = 1,...,n
```

Define the new bracket as follows:

```
     If F(XNEW) < F(X2)
     then
```

```
        X3 = X2
        X2 = XNEW
    else:
        X1 = XNEW
```

3.2.2 A New Method for Search Direction Specification

Glankwahmdee [25] and Glankwahmdee, Liebman and Hogg [26] defined an integer direction as follows:

Let V be a n-vector representing a direction in n-space.

Let the relative direction vector DR be as follows:

$$DR(i) = V(i)/B$$

where: $B = MIN \{ |V(i)| : i = 1,\dots,n\}$

Finally, let the integer direction M be:

$$M(i) = \langle DR(i) \rangle$$

where $\langle x \rangle$ is the nearest integer to x.

For example:

$$V = (1.7 , .5)$$

$$DR = (3.4 , 1)$$

$$M = (3 , 1)$$

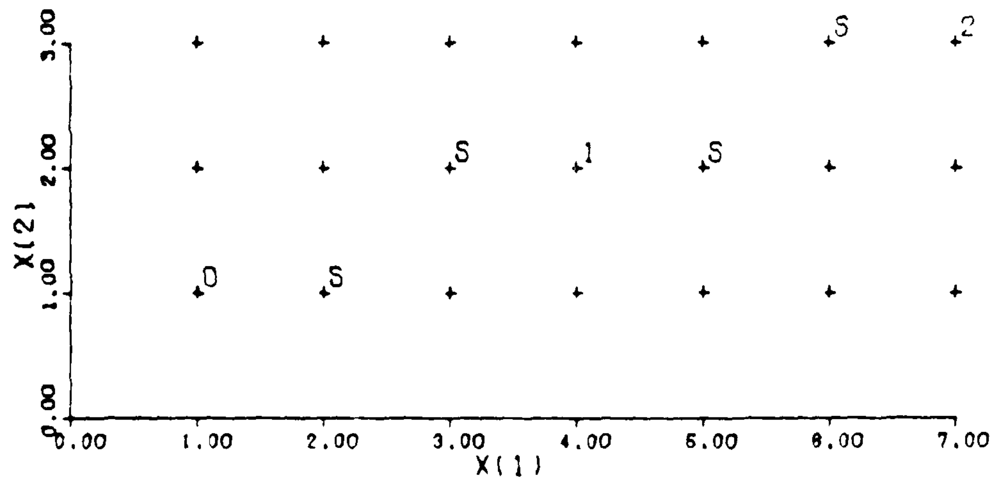
Thus a direction vector is scaled to give a minimum component of 1 and has all integer components. Points on a line from a base point XB in the integer direction M may be represented:

$P = XB + Y * M$

If the coordinates of XB are all integer, then points on the line have all integer components for integer values of the scaler Y.

In addition, Glankwahmdee defines a subsequential search interval in order to locate integer points near the line from the base point in the integer direction but falling between successive integer values of Y. These points are illustrated in Figure 3-1.

The integer direction as defined by Glankwahmdee has the advantage of providing evenly spaced discrete points on a line which makes the direction suitable for an efficient integer search technique based on Fibonacci numbers. Two disadvantages are that the discrete points may be widely spaced, necessitating a second type of search within a subsequential search interval, and that the integer direction may diverge widely from the original search direction. This second concept is illustrated in Figure 3-1. The second discrete point on the integer direction is (7,3), whereas when x(2) takes on value 3 along the original direction, x(1) would have value 7.8; and thus, the point (8,3) is closer to the original direction than is the point (7,3). The further along the integer direction a search proceeds, the more widely the original direction and the integer direction diverge.



O - BASE POINT
1 - POINT ON INTEGER DIRECTION (3,1) FOR Y=1
2 - POINT ON INTEGER DIRECTION (3,1) FOR Y=2
S - POINTS ON SUBSEQUENTIAL SEARCH
    INTERVAL ABOUT POINT 1

FIGURE 3-1. POINTS ON INTEGER DIRECTION AND
IN SUBSEQUENTIAL SEARCH INTERVAL

A third disadvantage of the integer direction is that, by the definition, certain directions are forbidden. Because the elements of the integer direction vector are integer and the smallest element is equal to one, then in two dimensions, for example, no direction between (1 , 1) and (1 , 2) is defined. Likewise no direction between (1 , 1) and (2 , 1) is defined. These forbidden directions may encompass a substantial portion of the parameter space, as illustrated in Figure 3-2. Within the 12 by 12 grid illustrated, 52 of the 144 points are inaccessable by any single unidimensional search when the integer direction is used.

As an alternative to the integer direction and subsequential search interval, the following is used in this research. Consider a normalized direction vector S and a base point XB. For points P where:

$$P = XB + Y * S$$

Let:

$$IP(i) = \langle P(i) \rangle \qquad \text{For } i=1,...,n$$

Thus, for any scaler Y one finds a point IP that is the discrete point nearest in each coordinate to the line from the base point along the given direction. For a segment of the line from XB along direction S, there are an infinite number of scalers Y and a finite number of nearest discrete points. In order to insure that unique discrete points are determined, it is sufficient to increment Y by an amount:

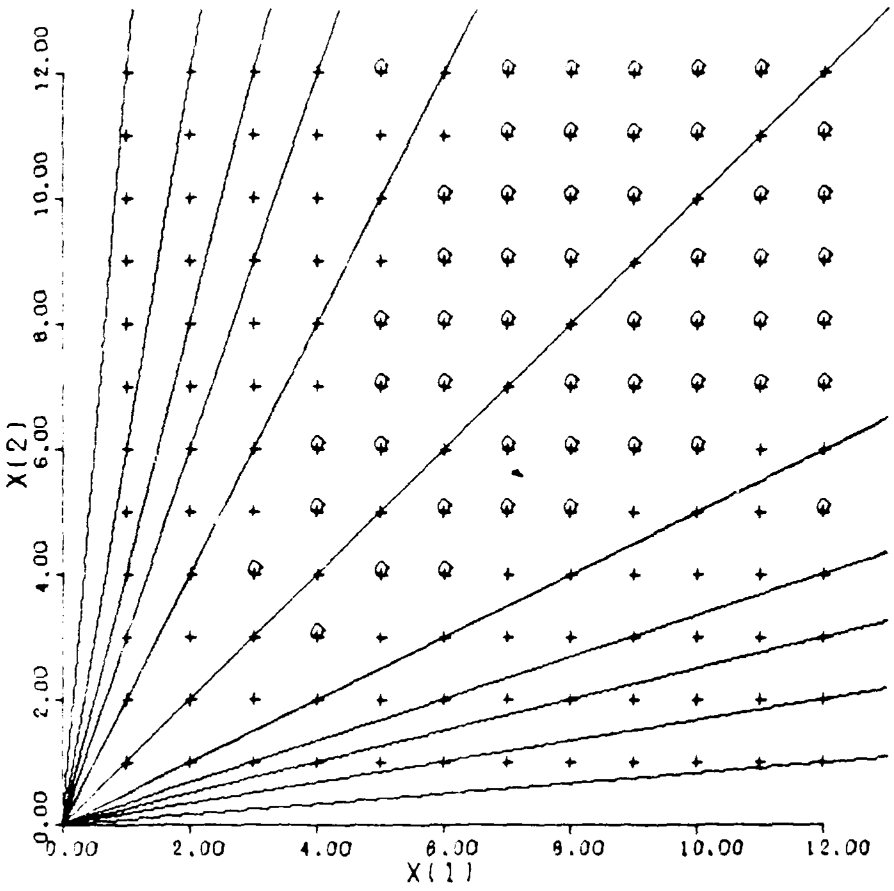


FIGURE 3-2. FORBIDDEN POINTS
(INDICATED BY ♀)

$b = 1/a$

where:

$a = \text{MAX} \{ |S(i)| : i=1,\ldots,n \}.$

We now have a formulation that allows a single search method to replace both the search along the integer direction and the subsequential search.

In addition, this formulation allows for searching over mixed integer spaces. Consider a normalized search direction S and a base point XB in a mixed integer space. Suppose the first m coordinates of the space are integer while coordinates m+1 to n are continuous. Then, as before, let:

$P = XB + Y * S$

$IP(i) = \langle P(i) \rangle \quad \text{for } i=1,\ldots,m$

$IP(i) = P(i) \quad \text{for } i = m+1,\ldots,n$

Thus the point IP is the nearest grid point to the point P in the mixed integer space.

3.2.3 A Modified Method for Handling Constraints

As noted in the previous chapter, penalty function methods for constraints pose several difficulties. Because of these difficulties, this research uses an explicit method of handling constraints adapted from the complex algorithm [5].

Nelder and Mead [46] suggested that explicit constraints, those expressing lower and upper bounds on variables, could be treated in their algorithm by specifying a very poor objective function value for any vertex that violates those bounds. Box [5], in the complex algorithm, handled implicit constraints by specifying retraction toward the feasible region for any point violating the constraints. Guin [29] in his proposed modification to the complex algorithm, suggested that the retraction rule be used for both implicit and explicit constraint violations. The ideas of specifying a very poor objective function value for infeasible points and of retraction toward the centroid for infeasible points may be advantageously combined. By incorporating a barrier function (a function with large values for infeasible points) into the objective function, the logic to provide for retraction of infeasible points may be eliminated from the algorithm. In particular, when a unidimensional search is incorporated into the algorithm, as discussed above, the behavior of the complex method, with barriers added to the objective function, is very similar to using the complex retraction logic. If the search begins at a feasible point and an infeasible point is subsequently located during the unidimensional search, then the search will automatically retract to a feasible point because of the poor objective function value of the infeasible point. If the search begins at an infeasible point and a feasible point is then located, the infeasible point will be rejected in favor of the feasible point on the basis of objective function value. Finally, if the search begins at an infeasible point and no feasible point is located, then an additional modification is required.

In addition to a barrier, the effective objective function used in this research includes a term that is proportional to the sum of constraint violations. If a unidimensional search begins at an infeasible point and no feasible point is located, then the search will select the point that has the minimum sum of constraint violations. Minimizing the sum of constraint violations is a classical heuristic for locating feasible points. If the initial complex contains infeasible points, then the effective objective function to be minimized, when those points are selected as base points for searches, is the sum of the constraint violations. This method allows the algorithm to proceed automatically from finding a feasible point to finding an optimal point.

The effective objective function minimized by the algorithm is illustrated, in a single dimension, in Figure 3-3. Inside the feasible region the effective objective function is simply the objective function of the problem being optimized. At the boundary of the feasible region the effective objective function is M where M is a large number with respect to the objective function value in the feasible

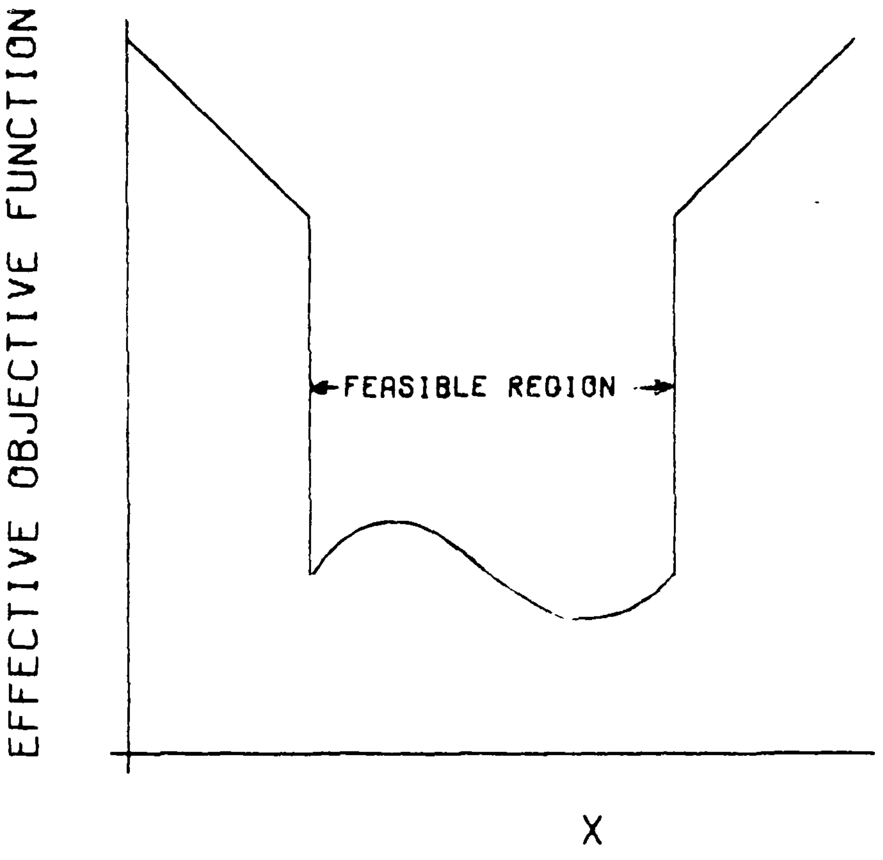


FIGURE 3-3. EFFECTIVE OBJECTIVE FUNCTION

region. Outside the feasible region the effective objective function is M + (violations) where M is as above and (violations) is the sum of the constraints violated.

The effective objective function in the infeasible region may be visualized as a funnel sloping towards the feasible region. The feasible region may be thought of as a well into which the search will fall when begun from outside the feasible region. The boundary of the feasible region may be considered a wall from which the search will rebound when begun from inside the feasible region.

One further point deserves attention. If a unidimensional search begins at, or locates, a feasible point and subsequently locates a point at which a constraint is violated, the evaluation of other constraints is immaterial. That is, any constraint violation at all is sufficient to assign a poor objective function value. The opportunity therefore exists to reduce computational effort on those searches that start from or have already found a feasible point, by evaluating the constraints one at a time, and stopping the evaluations as soon as an infeasiblity is discovered. In fact, given that the search has started at or found a feasible point, the evaluation of the effective objective function may be done in three stages. First, the point is checked against the upper and lower bounds for the variables. If any bounds are violated, proceed no further and assign the objective function a very poor value.

Second, the constraints are evaluated one at a time until a violation is found or all are evaluated. If a violation is found, proceed no further and assign a very poor value to the objective function. Finally, if the point has been determined to be feasible, compute the objective function value.

It is clear that the effective objective function is discontinuous at the boundary of the feasible region. As is noted by Lasdon et. al. [39], a unidimensional search procedure based on polynomial approximaticn cannot be expected to perform well on this type of objective function. In this research, a golden section search is used because of its insensitivity to discontinuties. Also, it is relatively efficient and offers flexibility in specifying termination critera.

3.2.4 Termination Criteria for the Unidimensional Search Component

Using the unidimensional search modification of the complex method removes the need to select reflection and contraction factors. But, there is a new question of how accurately to locate the optimal point in any given search direction.

The unidimensional search is terminated when one of two criteria is satisfied. If a specified number of search iterations have been performed, then the search is halted. If the interval of uncertainty is reduced to the point that further localization of the minimum is insignificant, then the search is terminated. What constitutes an insignificant change in the location of the minimum of the function along a direction is determined as follows: Suppose the increments (pseudo-increments in the case of continuous variables) of the variables are I(i) , i = 1,...,n. Along a direction D, the points to be searched may be represented as:

P = XO + S * D

where:

S is the stepsize

The smallest stepsize of interest is that which will result in a change of one increment along one of the coordinate axes. This step size (SMIN) may be calculated as:

SMIN = MIN {I(i) / D(i) : i = 1,...,n}

where:

I(i) is the increment for the ith variable

D(i) is the ith component of the direction vector for the unidimensional search

A stepsize smaller than SMIN is declared to be insignificant. Thus, the unidimensional search halts when the interval of uncertainty is a fixed percentage of the original interval (when the search is halted after a fixed number of iterations) or when the interval of uncertainty is no larger than the increment (or pseudo-increment) in any coordinate direction.

3.2.5 A New Algorithm Termination Criterion

A variety of termination criteria have been applied to the complex algorithm. Generally the algorithm is terminated when the variation in the objective function values of the vertexes in the complex is "small" or, alternatively, when the vertexes of the complex are "close" together. The first of these criteria can be quite problem dependent due to wide variation in the scaling of objective functions. The second is inappropriate when the parameter space is discrete with different increments assigned to the various coordinates. Here a measure based on average distance between vertexes can be misleading unless distances are normalized by the coordinate increments.

The termination criterion used in this research is based on specifying what comprises a significant variation of each variable. For discrete variables, a significant variation of the variable is taken as the increment for that variable. For each continuous variable, a pseudo-increment must be specified. For the actual termination test the size of the complex in each coordinate direction is determined. The size is calculated as follows:

Let:

$a(i) = \text{MAX} \{ V(j,i) : j = 1,\ldots,nv\}$

$b(i) = \text{MIN} \{ V(j,i) : j = 1,\ldots,nv\}$

Where: nv is the number of vertexes in the complex

then the size s(i) in coordinate i is:

$s(i) = a(i) - b(i)$

In testing for termination, the size of the current complex along each coordinate axis is compared to the increment (pseudo-increment in the case of continuous variables) for the respective coordinates. The number of coordinates where the extent of the current complex is less than the respective increments size is summed. If this sum is greater than a specified number (between 1 and n) then the termination criterion is satisfied.

3.3 Auxiliary Techniques

3.3.1 Regeneration Methods

Two types of regeneration have been implemented. The first uses alternate search directions and the second contracts the entire complex towards the best vertex. These methods were suggested by Glankwahmdee [25] and Nelder and Mead [46] respectively.

The alternate direction regeneration defines a search direction from the second worst vertex to the centroid of the remaining vertexes and performs a search along that direction. If the search is successful, the worst vertex is replaced and regeneration is terminated. If the search fails, then the third worst vertex is used, the fourth worst, etc., until all vertexes have been used. If no search is successful then the alternate direction regeneration is considered to have failed.

The contraction method of regeneration defines a new complex by moving every vertex one third of the way toward the best vertex. The best vertex is of course unchanged by this transformation which may be described as follows:

Let:

V be the matrix of coordinates describing the current complex

where:

V(j,i) is the ith coordinate of the jth vertex

nv is the number of vertexes in the complex

b is the index of the best vertex

then:

V(j,i) = V(b,i) + .667 * (j,i) - V(b,i)) ,

rounded to the nearest grid point,

j = 1,...,nv , i = 1,...,n

All results in this research were obtained using both regeneration methods sequentially when required. That is, if regeneration was required then the alternate direction method was tried first. If the alternate direction regeneration failed then the contraction method was used.

3.3.2 New Acceleration Strategies

Another idea developed in the research is the use of trajectory analysis as an acceleration strategy. Two types of trajectories have been investigated, linear and quadratic. The aim of these acceleration stratagies is to identify an objective function valley and then search along that valley to improve the solution.

The method used for valley identification is to assume that the best vertexes of the complex will tend to be located at or near a valley. This assumption is justified because of another of the modifications to the complex algorithm developed in this research, the incorporation of the unidimensional search. When a unidimensional search direction is used on an objective function that contains a valley, unless the search direction is parallel to the valley, the point along the direction with the best objective function value will be a point near the bottom of the valley. Thus, the use of unidimensional search to locate the new vertexes of the complex will, if valleys are present, tend to locate points in those valleys.

3.3.2.1 Linear Trajectories

If two points are selected that appear to be at or near a valley, then the line between them may provide a direction along which acceleration is possible. In this research the line between two best vertexes of the current complex is used to define a linear trajectory.

3.3.2.2 Quadratic Trajectories

In general the valleys of an objective function surface are not straight but curved. Quadradic trajectory analysis attempts to identify a curving valley by fitting a quadratic curve to three points assumed to be at or near the valley. Generally the three best vertexes of the current complex are used to define the trajectory.

The reader should note that this is not an attempt to fit a quadratic curve to the objective function surface, a procedure which requires evaluating the objective function at $(n+2)(n+1)/2$ points. Instead, this is an extension of the linear trajectory search that allows for a curved trajectory. Just as in the case of the linear trajectory, a unidimensional search is used to locate points with improved objective function values along the trajectory. Normally, this procedure would require evaluating the objective function at $n(n+1)/2$ points, but, by considering just two variables at a time, three points suffice to define a trajectory.

In calculating a quadratic approximation to an objective function surface the roles of the dependent variable (the objective function) and the independent variables (the variables of the parameter space) are clearly delineated. In calculating a trajectory, this is not the case. One of the variables must be selected to play the role of the independent variable while the remaining variables are treated as dependent variables. Each dependent variable is, in turn, paired with the independent variable. In this two dimensional space $2(2+1)/2 = 3$ points are sufficient to define a quadratic trajectory. When this has been done for each of the $n-1$ dependent variables a unidimensional search can be used to examine new points defined by extrapolating along the trajectory. The variable k is selected to play the role of the independent variable as follows:

Suppose A, B, and C are the points to be used to define the trajectory, and that

$$F(A) > F(B) > F(C)$$

then the variable k must be selected so that either

$$A(k) > B(k) > C(k)$$

or

$$A(k) < B(k) < C(k)$$

If more than one variable satisfies this criterion, then one of these variables can be chosen arbitrarily to be the independent variable. The restriction on the selection of the independent variables insures that the objective function is decreasing as the independent variable decreases (the first case), or as the independent variable increases (the second case). In either case, the presumption is made that further decreases in the objective function value are possible if the independent variable is varied in the indicated direction.

In summary, one variable is chosen to play the role of the independent variable. By considering just one coordinate at a time, three points are sufficient to define a quadratic trajectory in each of the remaining variables. Finally, a unidimensional search is used to vary the independent variable and the objective function of the points on the trajectory thus defined is evaluated.

A quadratic trajectory is illustrated in the following two dimensional example. Suppose the three points chosen to define the trajectory are:

$$A = (3 , 3)$$

$$B = (2 , 1)$$

$C = (1, 2)$

The (i)th coordinate of the points P on the desired trajectory are defined by the equation:

$$P(i) = E(i) + F(i) * d + G(i) * d * (d + C(k) - B(k))$$

where:

d is the search variable

k is the subscript for the independent variable

The parameters E, F and G are defined as:

$$E(i) = C(i)$$

$$F(i) = \frac{B(i) - C(i)}{B(k) - C(k)}$$

$$G(i) = \frac{\frac{A(i) - C(i)}{A(k) - C(k)} - F(i)}{A(k) - B(k)}$$

Letting X(1) play the role of the independent variable and carrying out the computations for this example yields:

$E = (1, 2)$

$F = (1, -1)$

$G = (0, 1.5)$

Since $F(k)=F(1)=1$ and $G(k)=G(1)=0$ then it is clear that the search variable d is merely an offset of the independent variable such that for $d = 0$ the point defined by the equation above is point C. This allows point C, a point with known objective function value, to serve as the base point for the search. If a point with an unknown function value were used as the base point, then one additional function evaluation would be required on each search. For d = 1 (independent variable increased by 1 from the base point) we can calculate:

$$P = (2 , 1)$$

As expected, the quadratic trajectory goes through the point B. The quadratic trajectory is illustrated in Figure 3-4. Using d as the search parameter, a unidimensional search is used to search along the trajectory for the point with the best objective function value.

Note that, just as in linear acceleration directions, the values of the objective function at the points used to define the trajectory are not used in computing the trajectory. The points define a search space in a single variable (the search variable). The search space, instead of being a line, is a curve defined by the equations above. A unidimensional search is performed to locate the point along the curve with the best objective function value.

3.3.3 A New Decomposition Approach

Another idea explored is the use of a complex algorithm in conjunction with a decomposition of the optimization problem. For separable optimization problems the sectioning (one variable at a time) search is an efficient search technique. In problems with a large degree of interaction

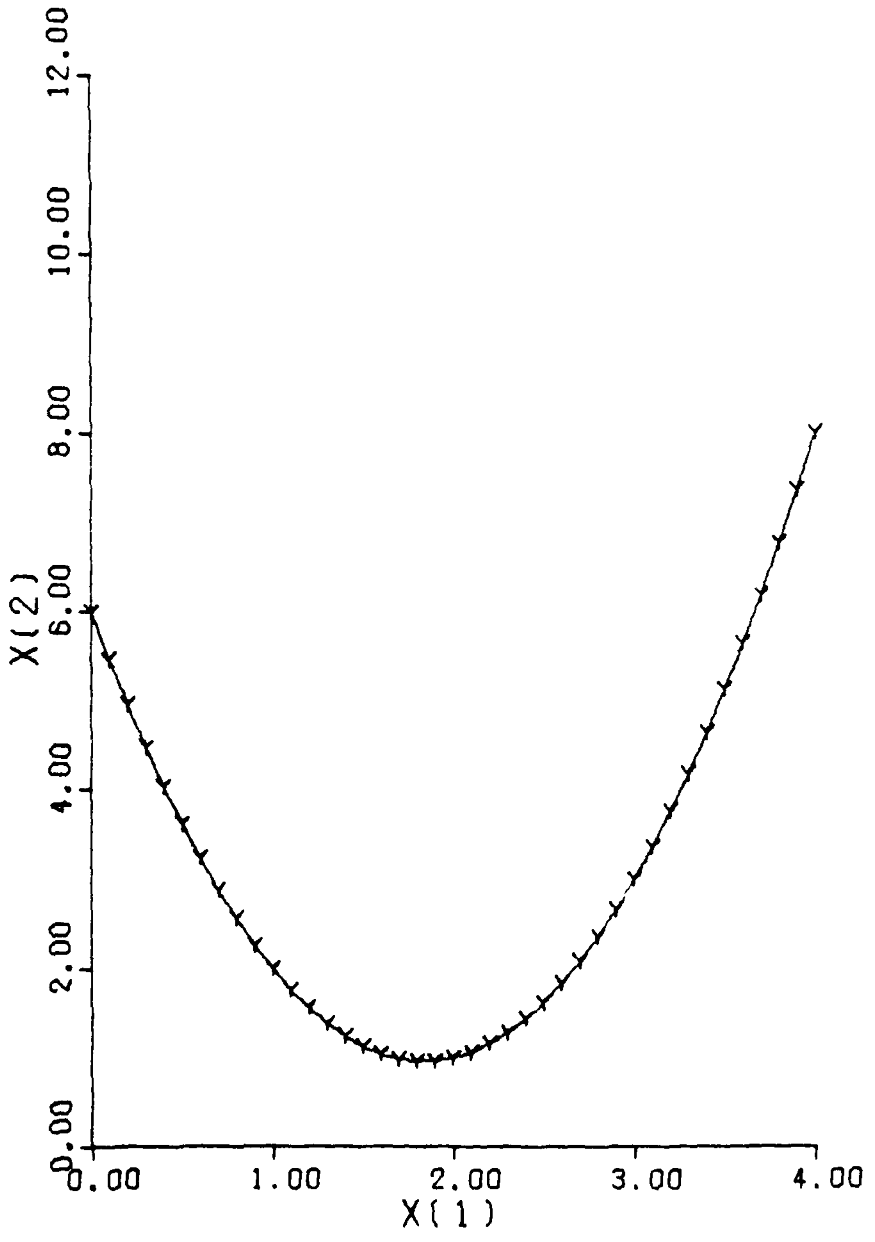


FIGURE 3-4. QUADRATIC TRAJECTORY

between the variables, however, sectioning search becomes inefficient, requiring a large number of iterations or perhaps failing completely [77].

For optimization problems in which subsets of variables interact but in which there is little or no interaction between the subsets of the variables, an extension of the sectioning search can be effective. Instead of a unidimensional search along a single coordinate axis, a complex search is made within a subspace defined by a subset of variables.

The complex algorithm is particularly well suited for this subspace search for two reasons. First, the complex method is reported to be more robust in spaces of low dimensionality [6]. Second, the subspace search may be easily incorporated into the complex algorithm without altering the logic implementing the algorithm. In order to restrict the search to a subspace it is only necessary to initialize the vertexes, defining the complex such that all vertexes lie in the subspace. Since each new vertex that enters the complex is a linear combination of vertexes in the current complex, the search is automatically restricted to the subspace. In order to expand the complex into the full space (or a different subspace) it is only necessary to redefine the coordinates of the vertexes of the complex. As long as this is done so that the coordinates of the vertex with the best objective value are unchanged, the overall algorithm will always move to points of decreasing objective function value.

3.3.4 A Grid Approach

A frequently suggested approach to optimization of functions is to select a grid on the parameter space and to calculate the objective function at every grid point. A grid of smaller increments is then constructed in the vicinity of the point with the best objective function value. The process is continued until the grid size has been reduced to a small size. In order to avoid an exorbitant number of function evaluations, particularly when the number of variables in the parameter space exceeds two or three, the initial grid must be very coarse. This increases the chances for failure of the method. In any case the exhaustive evaluation of the grid points for decreasing grid sizes results in an inefficient algorithm for all but the smallest problems.

Given a discrete optimizing algorithm, a more efficient alternative exists. Instead of computing the objective function for every grid point, a discrete algorithm can be used to locate the grid point with the best objective function value. This procedure has been suggested by Cella and Soosaar [9] and a version that used a discrete complex algorithm was implemented by Simmons and Pike [66]. This modified grid algorithm is available through the composite algorithm simply by redefining the grid size and restarting the discrete modified complex algorithm from the best point found when the previous grid was used.

3.4 Additional Search Algorithms

3.4.1 A Steepest Descent Algorithm

Glankwahmdee [25] and Glankwahmdee, Liebman and Hogg [26] defined an algorithm combining integer steepest descent and regeneration based on sectioning that was an efficient algorithm for unconstrained, integer optimization problems. In the composite algorithm Glankwahmdee's approach was incorporated, but an alternative method of specifing the discrete point along a search direction (described earlier in this chapter) was used. When combined with sectioning regeneration this implementation is refered to as SD/SECT. This option was included in the composite algorithm primarily for unconstrained problems, but it can be used for constrained problems when begining from an interior point before any constraints are encountered. The steepest descent option performs the following steps:

1. Calculate a gradient approximation at the current point.

2. Perform a unidimensional search along the direction of the gradient.

3. Update the current solution with the results of the search and return to step 1.

The gradient approximation G at a point X is calculated as follows:

$$G(i) = (F(X + Ri*I) - F(X)) / I(i)$$

if X + Ri*I is a feasible point

else:

$$G(i) = (F(X) - F(X - Ri*I)) / I(i)$$

if X - Ri*I is a feasible point

else:

$$G(i) = 0$$

where:

I is a column vector with

I(i) = increment for variable i or

the pseudo-increment for continuous variable i

Ri is a vector, the ith row of the identity matrix

3.4.2 Sectioning Algorithm

This algorithm varies one variable at a time by selecting each of the coordinate directions in turn and applying the unidimensional search procedure along each direction selected. This algorithm is included primarily for use as a regeneration method to be used in conjunction with the steepest descent search method.

3.5 Modified Complex Algorithm

The continuous modified complex algorithm (CMC) developed in this research may be described as follows:

1. Select algorithm parameters
   a. Number of vertexes for complex
   b. Increments for discrete variables
   c. Pseudo-increments for continuous variables
   d. Number of coordinates for termination

2. Initialize the coordinates of the vertexes for the initial complex and calculate the effective function value for each vertex.

3. Check the termination criteria; if satisfied go to step 10; otherwise, go to step 4.

4. Determine the vertex with the worst objective function value and the centroid of the remaining vertexes.

5. Perform a unidimensional search from the worst vertex in the direction of the centroid

6. If the objective function value resulting from the search is better than the objective function value of the worst vertex, then go to step 7; otherwise, go to step 8.

7. Replace the worst vertex with the result of the unidimensional search and go to step 3.

8. Apply the regeneration procedure.

9. If the regeneration procedure succeeded, go to step 3; otherwise, go to step 10.

10. Stop.

The modified complex algorithm can handle continuous variables in two ways, either treating them as continuous variables directly, or by a discrete approximation which treats the variables as discrete but with a small increment.

3.6 Initialization

The initialization required by the modified complex algorithm consists of selecting the vertexes of the initial complex. In this research a variation of the nonrandom starting complex [43] was used. Given an initial point X0 that satisfies the explicit but not necessarily the implicit constraints, the nonrandom starting complex consisting of 2n+1 vertexes is generated as follows:

Let:

$V(1,i) = X0(i)\ ,\ i = 1,\dots,n$

$V(k+1,i) = X0(i)\ ,\ i = 1,\dots,n\ ,\ i \neq k\ ,\ k = 1,\dots,n$

$V(k+1,k) = LB(k)\ ,\ k = 1,\dots,n$

where:

LB(k) is the lower bound for variable k

$V(n+k+1,i) = X0(i)\ ,\ i = 1,\dots,n\ ,\ i \neq k\ ,\ k = 1,\dots,n$

$V(n+k+1,k) = UB(k)\ ,\ k = 1,\dots,n$

where:

UB(k) is the upper bound for variable k

Some (or all) of the vertexes may be at infeasible points.

4 RESULTS

4.1 Introduction

In this chapter the results obtained by the algorithms developed in this research are analyzed. The robustness and efficiency of the algorithms are compared with algorithms previously reported in the literature.

One difficulty in testing an algorithm, such as the composite algorithm that provides so many opportunities for user intervention, is that what is to be tested is not a single algorithm but rather a multitude of algorithms, each defined by the user actions taken in the course of solving the problems. Because of this, the results reported here are primarly results for elements that comprise the composite algorithm. The major component of the composite algorithm developed in this research is the modifed complex algorithm. Accordingly the first four result sections report performance of this algorithm on four categories of problems. A fifth result section reports, by example, on some of the auxiliary techniques available in the composite algorithm. Hence, what is presented can not be a complete analysis of the performance of the composite algorithm, but is an attempt to convey some of the experience gained in the use of the composite algorithm.

4.1.1 Test Problems

The goal of this research was the development of a practical optimization algorithm applicable in engineering design. Accordingly, the majority of the test problems were selected to represent problems of this type. However, since few engineering design problems are unconstrained, additional unconstrained test problems have been selected from the literature. Some of these are functions which have been specifically designed to test features of unconstrained optimization algorithms and have become "classics" in the field of optimization.

For the purposes of clarity and reproducibility, the FORTRAN language subroutines used to compute the objective functions for the test problems (and constraint functions for constrained problems) are listed in Appendix 2. For those problems where the functions may be written as relatively simple mathematical expressions, these mathematical expressions are incorporated, along with the detailed research results, in Appendix 1.

4.1.2 Criteria for Evaluation

The algorithms tested in this research are compared both in robustness (the ability to find an optimal solution) and efficiency (speed of solution). With such a wide variety of test problems as have been used in this research, no single set of criteria for measuring robustness has been found to be satisfactory. Instead, for each category of results, a criterion has been selected which highlights the differences between the algorithms tested. These criteria are discussed at the beginning of each result section. In each case, a criterion has been selected which is meaningful in the context of engineering design optimization. For example, solutions that are close in objective function value are equal, for practical purposes. In general, a success criterion is specified, and the robustness of an algorithm is estimated by counting the number of problems for which the algorithm finds a solution which meets the criterion. For discrete problems, the number of times that algorithm finds the best solution is also considered. Efficiency is measured in number of function evaluations or, in the case of constrained problems, number of function and constraint evaluations.

4.1.3 Algorithms Used for Comparison

The generalized reduced gradient (GRG) algorithm is a widely used gradient based algorithm for constrained nonlinear problems. Ragsdell [55], after tests with 35 algorithms on 30 constrained non-linear problems, concluded that three GRG algorithms tested were sucessful on more problems and generally used less computer time than the other algorithms tested. Hence, the GRG algorithm represents a highly sucessful, efficient algorithm and will be used for comparison in order to evaluate the algorithm proposed in this research for continuous constrained problems. The particular GRG code used was prepared in 1975 by L. S. Lasdon at Case Westerm Reserve University.

The flexible tolerance algorithm (FLEX) developed by Paviani and Himmelblau [53] is a direct search algorithm for constrained, nonlinear problems. It uses a variation on the penalty function technique by varying the penalty parameter as the algorithm proceeds, rather than between cycles. A FORTRAN listing of the algorithm is given in an appendix to Himmelblau [32]. Two changes were made to the program as listed in the reference. Between card number 1340 and 1350 the FORTRAN statement:

```
      INF = I
```

was added. In subroutine FEASBL, the variable SIZE, which is undefined, is given the same value as the variable SIZE in the main program.

The Nelder and Mead algorithm (NM) is a direct search algorithm for unconstrained nonlinear problems. A FORTRAN listing of the program used to obtain the comparison results is included in Himmelblau [32].

Results obtained by Glankwahmdee [25], using a modification of the discrete complex algorithm (COMPLEX) suggested by Beveridge and Schechter [4], were used for comparison to the results obtained by the DMC algorithm.

An integer gradient, steepest descent and sectioning algorithm (G-2) developed by Glankwahmdee [25] was used for comparison purposes on some unconstrained discrete problems. The G-2 algorithm was the most successful of the algorithms developed by Glankwahmdee.

The G-2 algorithm, with penalty functions added to handle constraints, has been used for comparison on some constrained discrete problems. These results are labeled G-2/P.

Finally, a discrete solution was obtained by applying a rounding and N1 neighborhood search to the continuous solutions obtained by GRG on some constrained nonlinear problems. This algorithm is labeled GRG/R/N1.

4.1.4 Parameters

In order to provide a consistent and reportable comparison basis, it was necessary to arbitrarily fix the values of certain parameters. Unless otherwise specified the following parameter settings for the discrete modified complex (DMC) and continuous modified complex (CMC) algorithms were used to obtain the results cited in this research.

1. Number of function evaluations allowed on any one unidimensional search is 6.

2. Number of vertexes for the complex is 2n+1.

3. Number of coordinates collapsed for termination is n-1 for discrete problems and <(n+1)/4> for continuous problems.

These parameter values were selected on the basis of some preliminary exploratory work with the algorithm.

4.2 Results: Constrained Discrete Problems

The results discussed in this section are for constrained, discrete, nonlinear problems which were solved using three different algorithms: the generalized reduced gradient, rounding and neighborhood search algorithm (GRG/R/N1); the integer gradient, steepest descent with penalty functions algorithm (G-2/P); and the discrete modified complex algorithm (DMC).

4.2.1 Criteria for Evaluation

The results of the three algorithms are analyzed for robustness and efficiency. In evaluating robustness, the solutions obtained were put into one of three catagories. The first catagory is for the best solution obtained by any of the three algorithms. All other solutions are categorized as acceptable or unacceptable. The criterion for acceptable solutions which was used for the unconstrained discrete problems (based on the objective values of points in the N1 neighborhood of the best known solution) was unsuitable because in many cases the best known solution had few feasible points in its N1 neighborhood. However, the test problems represent engineering optimization problems in which the optimal objective functions did not have value zero. In these problems, the objective functions are expressed in terms of cost, weight, yield or other physical attributes. Thus, it is meaningful to measure differences in objective values as percentage deviations from the best known solution. The actual percentage deviation considered to be significant was arbitrarily taken to be one percent. The criterion for efficiency is the number of function and constraint evaluations.

4.2.2 Discussion

The results of the three algorithms on eleven engineering design problems (for a total of fifteen sample problems, since some alternative increment sizes for design variables were explored) are summarized in Table 4-1. In this table, the notation "*" indicates the best solution for any of the three algorithms. An "A" indicates an acceptable solution and an "X" indicates an unacceptable solution. The notation "X,NFS" indicates those cases where the algorithm failed to locate any feasible solution to a problem.

The DMC algorithm proved to be far more robust on these discrete constrained problems than were the other algorithms. The GRG/R/N1 solution was better than the DMC solution in only two of the fifteen examples and failed on seven examples. G-2/P was better than DMC on only one problem and failed on eleven problems, although three of these failures were on one problem, C-6. On problem C-6, which was run using three different increments, an

TABLE 4-1: Computational Results for Constrained Discrete Problems.

		GRG/R/N1		G-2/P		DMC		
PROBLEM	NUMBER OF VARIABLES/ CONSTRAINTS	SOLUTION QUALITY	NUMBER OF FUNCTION AND CONSTRAINT EVALUATIONS	SOLUTION QUALITY	NUMBER OF FUNCTION AND CONSTRAINT EVALUATIONS	SOLUTION QUALITY	NUMBER OF FUNCTION EVALUATIONS	NUMBER OF CONSTRAINT EVALUATIONS
C-2B	2/2	*	63	*	212	*	28	38
C-3C	2/1	*	99	X,NFS		*	87	178
C-3D	2/1	*	68	X	309	*	51	93
C-4B	5/6	A	66	X	38	*	[illegible]	69
C-4C	5/6	X	66	X	153	*	91	102
C-6B	3/2	X	75	X,NFS		*	79	114
C-6C	3/2	X	75	X,NFS		*	109	148
C-6D	3/2	*	75	X,NFS		A	169	220
C-7B	3/2	*	29	X	126	X	91	117
C-13B	5/7	X,NFS		X	299	*	424	496
C-17	2/1	X	94	A	159	A	53	86
C-18	2/1	*	77	A	118	*	49	68
C-19	2/4	*	68	X	82	*	21	30
C-20	6/5	X	1069	X	145	*	405	751
C-21	2/3	X,NFS		*	183	X	74	85
TOTAL *		7		2		11		
TOTAL X		7		11		2		

KEY: * - best solution, A - acceptable solution, X - unacceptable solution

infeasible starting point resulted in the failure of the penalty function algorithm to locate any feasible solutions.

The DMC algorithm proved to be generally more efficient as well. The number of function/constraint evaluations is about the same for GRG/R/N1 and DMC on those problems where both algorithms find acceptable solutions. For the few problems where G-2/P achieved acceptable solutions, DMC used only about 40 percent as many evaluations.

4.2.3 Details

In this section a problem by problem commentary on the relative performances of the algorithms will be presented. Again, only noteworthy or unusual circumstances will be discussed.

Problem C-2 is a model of a two bar plane truss in two variables and two constraints. On problem C-2B, all three algorithms find the same solution which is immediately adjacent to the continuous solution. Noteworthy here is that, with the variables treated as discrete, the DMC algorithm requires fewer function evaluations than does GRG in solving the continuous approximation.

Problem C-3 is a design model for a journal bearing in two variables and one constraint. Problem C-3C illustrates a difficulty with internal penalty function formulations.

From the infeasible starting point, no feasible solution could be found by G-2/P. Again, the optimal solution is adjacent to the continuous optimum. From a feasible starting point (Problem C-3D), the gradient based G-2/P method terminates at the grid point nearest the continuous optimum which is not the discrete optimum.

Problem C-4B, which is problem B due to Box [5], shows clearly a case in which the discrete optimal solution is far removed from the continuous optimum. In particular, the third design variable moves from its upper bound at the continuous optimum to the lower bound at the discrete optimum. However, when smaller increments are used for the design variables, the G-2/P algorithm correctly moves the third coordinate toward the lower bound, but does not find as good a solution as does DMC.

Problem C-6 is a model for design of a flywheel in three variables and two constraints. On Problem C-6 three alternative increments were tried. In both of the cases with larger increments the DMC algorithm finds a point at some distance from the continuous optimum and which had a better value than that found by GRG/R/N1. Only with the smallest increment does rounding yield the best solution. The failure of the interior penalty algorithm, G-2/P, is attributed to the infeasible starting point.

Problem C-7 is a version of the "post office problem" to maximize the volume of a rectangular shipping container subject to a constraint. The problem has three variables. Problem C-7 was selected to illustrate a problem which can occur when using the penalty formulation on discrete problems. In this case, the optimal discrete solution is the same as the continuous solution and the constraint is satisfied exactly at that point. Most penalty formulations can not yield the optimum because the constraint is exactly satisfied at the optimum and hence yields an infinitely large penalty. The penalty formulation used in G-2/P was modified in an attempt to overcome this difficulty, but G-2/P still failed to locate the optimum on this problem. The GRG/R/N1 yields the optimal solution (since no rounding was necessary). From the given starting point, DMC halts at a point near the optimal solution with objective function value within three percent of optimum.

Problem C-13 is an unpublished design model for a reinforced concrete bridge. This problem has five variables and seven constraints. Problem C-13 has a more complex constraint set than do most of the other problems. Since no feasible solution is found in the N1 neighborhood of the rounded continuous optimum, GRG/R/N1 fails. The penalty method also fails on this problem and terminates at a point with an excessively large objective function value. The DMC algorithm, however, finds a feasible solution with an objective function value within three percent of the continuous optimum.

Problem C-17 is a design model for a reinforced concrete beam with two variables and two constraints. On this problem all three algorithms found different solutions. The solutions are close to one another and within one percent in objective function value.

Problem C-18 is a modification of problem C-17. The cost coefficients in the objective are different which results in the optimal solution being at a different point. This problem is another case where the gradient based G-2/P algorithm located the same point as obtained by GRG/R/N1, while DMC located a different point with an insignificantly better (by .1 percent) objective function value.

Problem C-19 is a simple example problem for the design of a hatch cover. The problem has two variables, two constraints and a total of only 80 grid points within the bounds specified for the variables, with some of these points infeasible due to the constraints. The G-2/P algorithm evaluates 82 points and does not find the optimal solution. The source for this problem [24] describes an algorithm using penalty functions both for the constraints and for discretization. Their solution, which is the same as that found by DMC in 21 objective function and 30 constraint evaluations, required 641 objective function and 641 constraint evaluations.

Problem C-20 is a design model for a shell and tube condenser which has six variables and five constraints.

This problem yields three different solutions from the three algorithms. The DMC solution is clearly best; the G-2/P and GRG/R/N1 solutions are seven and ten percent worse, respectively.

Problem C-21 is a design model for a wooden frame. The problem has two variables and three constraints. This problem also had no feasible discrete solution in the N1 neighborhood of the rounded GRG solution causing GRG/R/N1 to fail. G-2/P found the optimal solution, but DMC did not Investigation revealed that one of the constraints paralleled the X(2) axis and that the complex collapsed against this constraint.

4.2.4 Conclusions

The DMC algorithm was clearly superior in robustness and efficiency to the other algorithms tested on these constrained discrete problems. The GRG/R/N1 failures are partially attributed to its propensity to locate local optimum near the starting point. Also, for problems where the discrete optimum is not located near the continuous optimum, this algorithm cannot be sucessful. The penalty method, G-2/P, failed on problems where an infeasible starting point was given. The use of the gradient to guide the search results in directing the search in the direction of the continuous optimal solution. In two of the above problems, G-2-P located solutions adjacent to the continuous solution when the discrete optimum lay elsewhere.

4.3 Results: Constrained Continuous Problems

In this section results are presented for the DMC algorithm used to approximate a continuous algorithm. This was done by treating all variables as discrete with a small increment (.001). The results of the DMC discrete approximation are compared to the Flexible Tolerance (FLEX) algorithm and the Generalized Reduced Gradiant (GRG) algorithm on sixteen constrained nonlinear problems.

4.3.1 Criteria for Evaluation

The algorithm results are analyzed to obtain measures of algorithm robustness and efficiency. Robustness is measured by counting the number of solutions obtained by each algorithm that meet a specified success criterion. An algorithm is credited with a success on a problem if the solution obtained has an objective function value within two percent of the best objective function value obtained by any of the three algorithms. Efficiency is measured in terms of the number of objective function and constraint evaluations.

4.3.2 Discussion

The results are summarized in Table 4-2. In this table the acceptable solutions are denoted "A" and the unacceptable solutions are denoted "X." By the stated criterion for robustness, the GRG algorithm had five failures; FLEX and DMC each had three failures. A comparison of relative efficiency achieved showed that FLEX and DMC use about the same number of objective function evaluations, but the number of constraint evaluations averages five times more for FLEX. Although the gradient based GRG algorithm was not as robust, its higher efficiency was illustrated by the fact that the DMC algorithm required fourteen times as many function evaluations.

4.3.3 Details

The following paragraphs include a problem by problem summary of the results obtained. Only noteworthy or unusual circumstances will be discussed.

Problem C-1 is a model of an alkylation process. The original model had ten variables and included three equality

TABLE 4-2: Computational Results for Constrained, Continuous Problems.

		GRG		FLEX			DMC		
PROBLEM	NUMBER OF VARIABLES CONSTRAINTS	SOLUTION QUALITY	NUMBER OF FUNCTION AND CONSTRAINT EVALUATIONS	SOLUTION QUALITY	NUMBER OF FUNCTION EVALUATIONS	NUMBER OF CONSTRAINT EVALUATIONS	SOLUTION QUALITY	NUMBER OF FUNCTION EVALUATIONS	NUMBER OF CONSTRAINT EVALUATIONS
C-1A	5/14	X	29	A	3161	146566	A	2151	2829
C-2A	2/2	A	57	A	493	2302	A	141	200
C-3A	2/1	A	63	X	321	2252	A	338	573
C-3B	2/1	A	94	A	210	933	A	195	294
C-4A	5/6	A	60	A	1720	50991	A	2180	2593
C-5	2/3	A	19	A	354	2740	A	142	212
C-6A	3/2	A	71	A	630	7808	A	530	704
C-7A	3/2	A	25	A	373	3796	A	417	512
C-8	2/6	X	89	A	272	6004	A	1815	19[illegible]
C-9A	5/10	A	113	A	395	9653	X	714	948
C-9B	5/10	A	95	A	812	17056	A	703	1076
C-10	5/6	A	236	A	684	21906	A	871	1035
C-11	6/4	X	109	A	3316	72997	A	3448	4756
C-12	3/14	X	134	A	372	4407	A	386	431
C-13A	5/7	A	130	X	408	13145	A	516	644
C-14	9/13	X	10	A	1701	[illegible]49	A	1563	1957
C-15	4/5	A	345	A	1125	12919	X	1388	1793
C-16	6/4	A	133	X	48	5353	X	328	598
TOTAL X		5		3			3		

KEY: A - acceptable solution, X - unacceptable solution constraints. The model was reformulated by solving the equality constraints for three of the ten variables resulting in a model with seven variables. The original eight inequality constraints of the model, plus the bounds on the variables removed by solving the equality constraints, resulted in a total of fourteen inequality constraints. Although the "optimal" solution provided in the source [3] is superior to that found here, it does not satisfy the constraints.

This problem illustrates a common difficulty with gradient based techniques which is that they tend to move very efficiently to a local optimal solution near the starting point. The direct search techniques, while less efficient in number of function evaluations, do not make a headlong plunge to the nearest local solution, but wander more about the solution space and thus stumble upon solutions further from the starting point.

An exorbitant number of constraint evaluations for the FLEX algorithm reveals the difficulty in locating feasible or near feasible solutions. The variables in this problem are closely interrelated by the constraint functions, so that adjusting the variables to satisfy one constraint results in violating another constraint which in turn requires further adjustment to the variables.

On problem C-3A, which has an infeasible start, GPG and DMC found essentially the same solution.

AD-A092 435 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH F/G 12/1
A COMPOSITE ALGORITHM FOR MIXED INTEGER CONSTRAINED NONLINEAR O--ETC(U)
JAN 80 D B FOX
UNCLASSIFIED AFIT-CI-80-1D NL

2 OF 3
AD
A092435

moves to a point distant from the starting point to attain near feasibility and never recovers. On variation B, with a feasible starting point, all three algorithms find essentially the same solution.

On problem C-6, while GRG and FLEX arrived at essentially the same solution, DMC found a solution with virtually the same objective function value, but at a vastly different point.

Problem C-8 is a chemical equilibrum problem that began with ten variables and three equality constraints. After solving the equality constraints, the resulting problem has seven variables and six inequality constraints. Each algorithm determines a quite different solution and GRG locates a point with an objective function value 2.2 percent worse than obtained by the other algorithms.

Problem C-9 has five variables and ten linear inequality constraints. The failure of the DMC algorithm on variation A could be anticipated because four of the five variables at the starting point lie on the boundary. Thus the non-random starting complex begins with five of the eleven vertexes at the same point. This redundancy results in premature termination at a suboptimal solution. From a starting point away from the boundry this difficulty does not occur. With that exception, all algorithms locate essentially the same solution.

Problem C-10 is a design model in five variables and six constraints. The "optimal" solution provided in the source [10] slightly violates two of the constraints. All three algorithms find essentially the same solution, which differs from that provided in the source.

Problem C-11 is a refinery heat integration problem in six variables and four constraints. Each algorithm finds quite different solutions, but FLEX and DMC find similar and slightly better objective function values.

Problem C-12 is a version of the alkylation process in problem C-1. In this version there are only three variables and seven constraints. FLEX and DMC find the same solution while GRG locates a local solution very close to the starting point which has an objective function value 25 percent worse than that obtained by the other algorithms.

Problem C-13 has five variables and seven constraints. All three algorithms find quite different solutions, with the GRG and DMC solutions superior and close in objective function value.

Problem C-14 has nine variables and thirteen constraints. GRG makes no progress from the given starting point. FLEX and DMC find solutions with similar objective function values.

Problem C-15 is a model for design of a welded structure with four variables and five constraints. Here GRG and FLEX find essentially the same solution while DMC terminates at the non-optimal solution. In this case, the complex flattens against a constraint and the algorithm terminates prematurely. Restarting the algorithm from this point resulted in finding essentially the same solution as the other two algorithms.

Problem C-16 has six variables and three constraints. The optimal solution provided in the reference slightly violates one constraint. FLEX is unable to locate near-feasible points even though several different values for the initial tolerance criteria were tried. The poor solution obtained by DMC is partially due to too large a stepsize. Better solutions were obtained with smaller stepsizes.

4.3.4 Conclusions

In summary, on these constrained continuous variable problems, the direct search algorithms FLEX and DMC were more robust, but less efficient than GRG. The GRG failures are primarily due to the algorithm locating local optima near the starting points. Conversly, the robustness of the DMC algorithm is due, at least in part, to its ability to locate local optima other than those near the starting point. Of the three failures of the DMC algorithm, one (C-9) was predictable because of the inappropriate starting point.

While FLEX and DMC used about the same number of function evaluations, FLEX used about five times as many constraint evaluations.

Of course any serious attempt to obtain solutions to these problems would use several starting points. This would be likely to improve the robustness of all of the algorithms but would particularly benefit the GRG algorithm because of the tendency, discussed earlier, for gradient-based methods to locate the nearest local optima. Under these conditions the algorithms would be more equal in robustness and the greater efficiency of the GRG algorithm would be a decided factor in its overall superiority.

4.4 Results: Unconstrained Discrete Problems

The discrete modified complex (DMC) algorithm was used to solve some all integer, unconstrained problems used by Glankwahmdee [25]. The results of using the DMC algorithm are compared to those reported for COMPLEX and G-2. Six additional problems, which were not used by Glankwahmdee, and which represent engineering design applications, were solved with the G-2 algorithm as well as the related SD/SECT algorithm. The results are compared to those obtained using DMC.

4.4.1 Criteria for Evaluation

The robustness of the algorithms in this section is measured by counting the number of problems for which the algorithm locates an acceptable solution. Since several of the problems have minimum objective function values of zero, a success criterion based on the percentage deviation from the optimal value is inappropriate. For these problems, an algorithm is credited with a success if it locates a solution with an objective function value less than the median of the objective function value for the points in the N1 neighborhood of the optimal solution. As usual, efficiency is measured in number of objective function evaluations.

4.4.2 Discussion

The results of the COMPLEX, G-2 and DMC algorithms on the problems used by Glankwahmdee are shown in Table 4-3. The most robust algorithm as measured either by the largest number of optimal solutions or by the smallest number of unacceptable solutions is G-2. The DMC algorithm is a close second and represents considerable improvement over COMPLEX.

The only DMC failure (problem U-8A) arose from the combination of a starting point virtually centered in the

TABLE 4-3: Computational Results for Unconstrained, Discrete Problems.

		COMPLEX		G-2		DMC	
PROBLEM	NUMBER OF VARIABLES	SOLUTION QUALITY	NUMBER OF FUNCTION EVALUATIONS	SOLUTION QUALITY	NUMBER OF FUNCTION EVALUATIONS	SOLUTION QUALITY	NUMBER OF FUNCTION EVALUATIONS
U-1A	2	A	47	*	61	*	96
U-1B	2	X	57	*	51	*	89
U-2A	2	A	43	A	25	A	91
U-2B	2	X	37	A	19	A	95
U-2C	2	*	67	*	39	*	55
U-3A	2	*	40	*	38	*	89
U-4A	2	A	134	A	60	A	95
U-4B	2	A	70	A	100	*	145
U-4C	2	A	50	A	44	A	117
U-5A	2	*	44	*	43	*	83
U-6A	4	X	215	*	140	A	226
U-7A	4	*	220	*	112	A	215
U-8A	5	X	200	A	360	X	94
U-8B	5	A	368	A	463	A	724
TOTAL *		4		7		6	
TOTAL X		4		0		1	

KEY: * - best solution, A - acceptable solution, X - unacceptable solution

region specified by the upper and lower bounds of the variables and the use of the nonrandom starting complex. The symmetry of the initial complex under these conditions results in search directions along which no objective function improvement is possible. Only one move was made and the algorithm terminated at a point immediatly adjacent the starting point. It is noteworthy that use of a random starting complex resulted in finding acceptable solutions in three successive trials (using different random number sequences). An acceptable solution was also found when an alternative starting point was used with the nonrandom starting complex (problem U-8B).

The increase in robustness of the DMC algorithm over the COMPLEX algorithm is achieved at the expense of more function evaluations. For those problems where both COMPLEX and DMC achieved at least an acceptable solution, DMC used about half again as many function evaluations. A comparison between G-2 and DMC shows that DMC uses about twice as many function evaluations.

The results of the G-2, DMC and SD/SECT algorithms on the six engineering design problems are summarized in Table 4-4. No unacceptable solutions were found by any of the three algorithms, however, G-2 found fewer optimal solutions. On three of the problems, G-2 terminated at solutions near the optimal solution but did not quite reach the optimum. G-2 and SD/SECT used about the same number of function evaluations while DMC used about one third more.

TABLE 4-4: Computational Results for Unconstrained Discrete Design Problems.

		G-2		SD/SECT		DMC	
PROBLEM	NUMBER OF VARIABLES	SOLUTION QUALITY	NUMBER OF FUNCTION EVALUATIONS	SOLUTION QUALITY	NUMBER OF FUNCTION EVALUATIONS	SOLUTION QUALITY	NUMBER OF FUNCTION EVALUATIONS
U-9B	2	*	61	*	94	*	143
U-10	3	A	37	*	64	*	146
U-11	3	A	90	*	66	*	130
U-12	2	*	37	*	49	*	186
U-13B	2	*	50	*	38	A	104
U-14	3	A	123	*	86	*	105
TOTAL *		3		6		5	
TOTAL X		0		0		0	

KEY: * - best solution, A - acceptable solution, X - unacceptable solution

4.4.3 Conclusions

The DMC algorithm is about as robust but is less efficient than the G-2 algorithm on these unconstrained discrete problems. The SD/SECT algorithm located the best solution more often than did G-2, which often halted at a nearby suboptimal point. As previously discussed, the ability to locate optimal solutions away from the starting point may make DMC a desirable algorithm to use on some unconstrained discrete problems.

4.5 Results: Unconstrained Continuous Problems

A selection of unconstrained problems in continuous variables were solved using the CMC, NM, and DMC algorithms. The DMC algorithm was used in the discrete approximation (increment .001 for each variable).

4.5.1 Criteria for Evaluation

All six problems in this section have objective function minima of zero. Because of this, a measure of robustness based on a percentage deviation of the objective function from the optimal is not meaningful. The robustness criterion used for these problems is based on the number of problems for which an algorithm finds acceptable solutions. An acceptable solution is defined as a solution with an objective function value less than .001. Efficiency is measured by the number of objective function evaluations.

4.5.2 Discussion

When the continuous modified complex (CMC) algorithm was used, disappointing results were obtained. The algorithm often failed. After making some progress toward the optimal solutions, the complex collapsed; and the algorithm terminated without reaching the optimal solution. However, when the normal reflection, expansion and retraction was used in place of the unidimensional search, as in the Nelder and Mead algorithm [46], then optimal solutions were usually found. The results of the CMC algorithm and the Nelder and Mead (NM) algorithm are compared in the first 2 columns of Table 4-5.

More detailed examination of the operation of the modified algorithm revealed that the unidimensional search

TABLE 4-5: Computational Results for Unconstrained, Continuous Problems.

		CMC		NM		DMC	
PROBLEM	NUMBER OF VARIABLES	SOLUTION QUALITY	NUMBER OF FUNCTION EVALUATIONS	SOLUTION QUALITY	NUMBER OF FUNCTION EVALUATIONS	SOLUTION QUALITY	NUMBER OF FUNCTION EVALUATIONS
U-2D	2	A	198	A	174	A	236
U-4D	2	X	1599	A	546	A	1137
U-4E	2	A	415	A	398	A	432
U-6B	4	X	1930	A	704	A	2175
U-7B	4	X	1271	A	1809	A	4211
U-15	3	X	936	A	365	A	1398
TOTAL X		4		0		0	

KEY: A - acceptable solution, X - unacceptable solution increased the tendency of the complex to collapse into a subspace. This collapse was the result of successive unidimensional searches locating new vertexes of the complex along a nearly straight portion of an objective function valley. This, in turn, led to premature termination of the algorithm at suboptimal solutions due to an inability to further improve the objective function within the subspace defined by the complex. Since all new points entering the complex are linear combinations of points in the current complex, then if the complex is within a subspace, no number of iterations can locate solutions outside that subspace.

As an illustration of this consider problem U-4D (Rosenbrock's function) which terminates after 1599 function evaluations. After the first 415 function evaluations the coordinates of the five vertexes of the complex have a .99991 correlation coefficient. This means that for any of the five points in the complex, the second coordinate value can be predicted from the first with accuracy to two decimal places. After 840 function evaluations the correlation has increased to .99998. This implies that the second coordinate of any point in the complex can be predicted from the first coordinate to four decimal places. Thus, the complex has effectively collapsed into a subspace, in this case a line. The remaining iterations were spent in a search along this line which did not contain the optimal solution for the problem.

These results would have been discouraging but for another discovery. When the variables are treated as discrete, as in the discrete modified complex (DMC) algorithm, even if in small increments, rather than continuous, the tendency of the complex to collapse into a subspace is counteracted. This can be explained as follows. When linear combinations of points in the complex are rounded to the nearest discrete point, points outside the subspace can be located. This process is illustrated in Figure 4-1. Suppose that all points in the complex lie in a subspace defined by the line from (0. , 0.) to (4. , 3.) A discrete search along that line might yield, for example, the point (2. , 1.), which does not lie within the original subspace. Once a point outside the subspace has entered the complex, other points outside the subspace can be represented by linear combinations of these points.

The results of the DMC algorithm are shown in the last column of Table 4-5. Compared to the Nelder and Mead algorithm, DMC is as robust but requires more function evaluations. On the average, DMC uses more than twice as many function evaluations as does the NM algorithm.

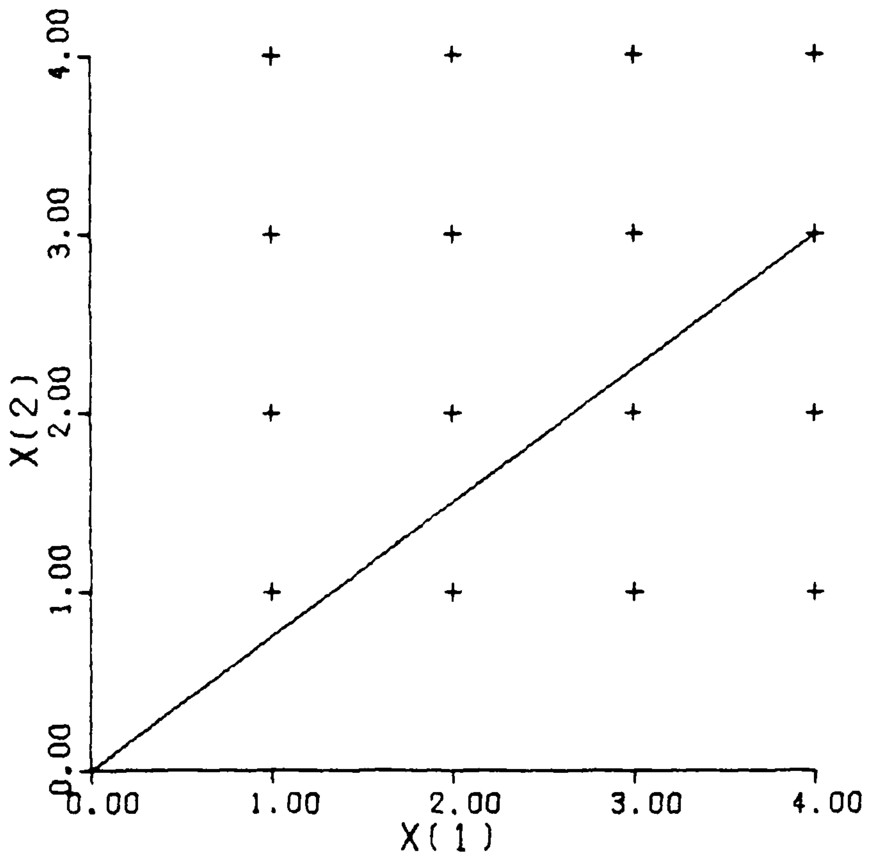


FIGURE 4-1. INTRODUCTION OF POINTS EXTERNAL TO A SUBSPACE

4.5.3 Conclusions

Overall these results indicate that the DMC algorithm is about as robust, but less efficient than the Nelder and Mead algorithm for continuous variable, unconstrained problems. On these problems, the CMC algorithm is unreliable.

4.6 Results: Composite Algorithm Options

In this section some sample results of the composite algorithm will be presented. These results are intended merely to indicate the potential of the composite approach, but not to analyze comprehensively the performance of all of the composite options. The examples in this section are illustrative of four composite algorithm options: (1) use of rounded continuous solutions as starting points, (2) decomposition by searching subspaces, (3) a grid approach using discrete search, and (4) trajectory analysis, which may be linear or quadratic.

4.6.1 Rounded Continuous Starting Points

In solving discrete or mixed discrete problems it is often fruitful, for those problems which can be solved as continuous variable problems, to look for optimal discrete solutions in the vicinity of the continuous optimum. While there can be no assurance that a better solution does not exist elsewhere, the number of successes attributed to the GRG/R/N1 algorithm on constrained discrete problems suggests that solutions worthy of a design engineer's consideration, although perhaps not optimal, may be located in the vicinity of the continuous optimum. As noted previously, a major difficulty in the case of complicated constraint sets may be locating a feasible point near the continuous optimum. A method for searching the vicinity of the continuous optimum is to round the continuous optimal solution to the nearest grid point and to use this point, which may be infeasible, as a starting point for the DMC algorithm. This may be done because the effective objective function formulation used in this research allows the search to proceed over infeasible as well as feasible points.

When this technique is applied to problem C-20 the solution obtained has an objective function value of 1168.028 after 315 objective function evaluations and 377 constraint evaluations. An additional 1056 objective function and constraint evaluations were required by GRG to locate the initial continuous solution. This discrete solution, which is in the vicinity of the continuous solution, has an objective function value five percent better than the best solution previously located.

This technique is, of course, a heuristic and there is no guarantee that an optimal solution will be found. For example, when this technique is applied to problem C-13 the solution obtained has objective function value 49306.4 after 168 function and 209 constraint evaluations. Note that an additional 136 objective function and constraint evaluations were required by GRG to obtain the original continuous solution. This solution is four percent worse in objective function value than is the solution obtained by using DMC alone. In this case the DMC search locates a superior solution that is remote from the continuous optimal solution.

4.6.2 Decomposition by Subspace Search

The decomposition strategy, discussed under auxiliary techniques in Chapter 3, was tried on problem U-16. The objective function for this problem is constructed such that the six variables are in two groups. The first three variables interact with each other as do the last three but there is no interaction between the two groups. The results reported below were obtained by alternately searching the subspaces defined by the first three and the last three variables. Twenty DMC iterations were used in each subspace search.

The results are summarized in the plot in Figure 4-2 which shows the value of the objective function vs the number of function evaluations for both the DMC algorithm and the decomposition method. The decomposition approach is much more efficient on this problem.

Problem U-17 is a variation on problem U-16 with a mild degree of interaction between the two groups of variables. The results, which are illustrated in Figure 4-3, are similar to those for problem U-16 with the decomposition method again being more efficient.

Problem U-18 is another variation on problem U-16 with an even greater degree of interaction between variables than in problem U-17. Figure 4-4 compares the results of the decomposition approach and the DMC algorithm. The interaction present is still mild and the decompostion algorithm is still more efficient than the DMC algorithm alone.

Figure 4-5 illustrates the fact that the efficiency of the decomposition approach is related to the degree of interaction between the groups of variables. The plot compares the progress of the decomposition algorithm on problems U-16, U-17 and U-18. Problem U-16, with the least interaction, is solved more efficiently than either of the other problems while problem U-18, with the most interaction is solved less efficiently than the other two problems.

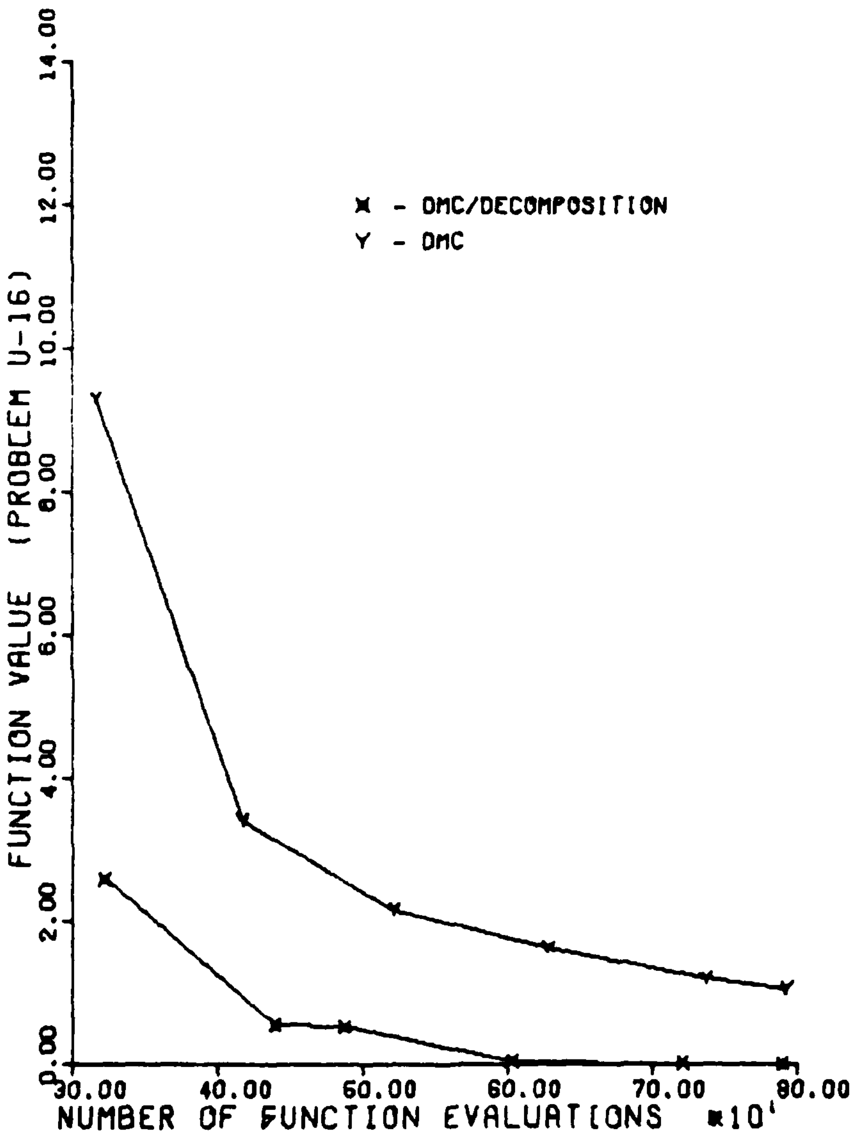


FIGURE 4-2. CHANGE IN FUNCTION VALUE VS NUMBER OF FUNCTION EVALUATIONS PROBLEM U-16

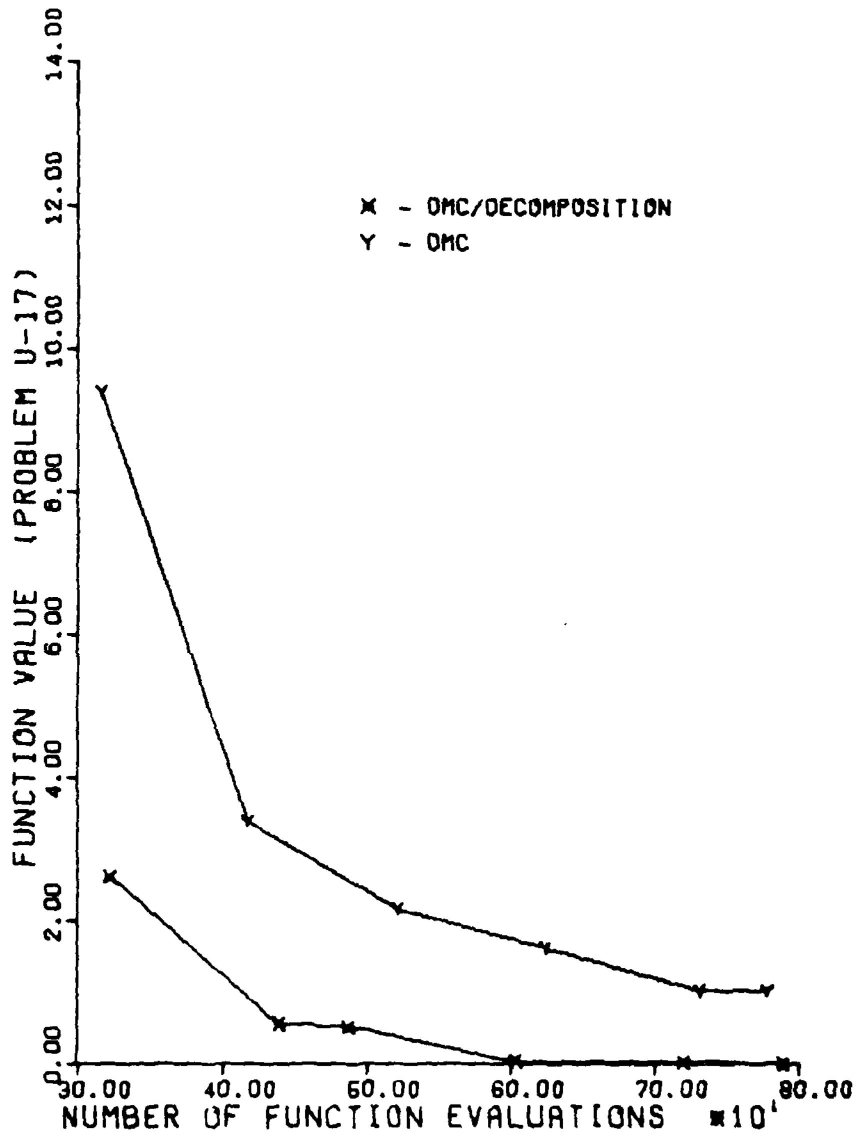


FIGURE 4-3. CHANGE IN FUNCTION VALUE VS
NUMBER OF FUNCTION EVALUATIONS
PROBLEM U-17

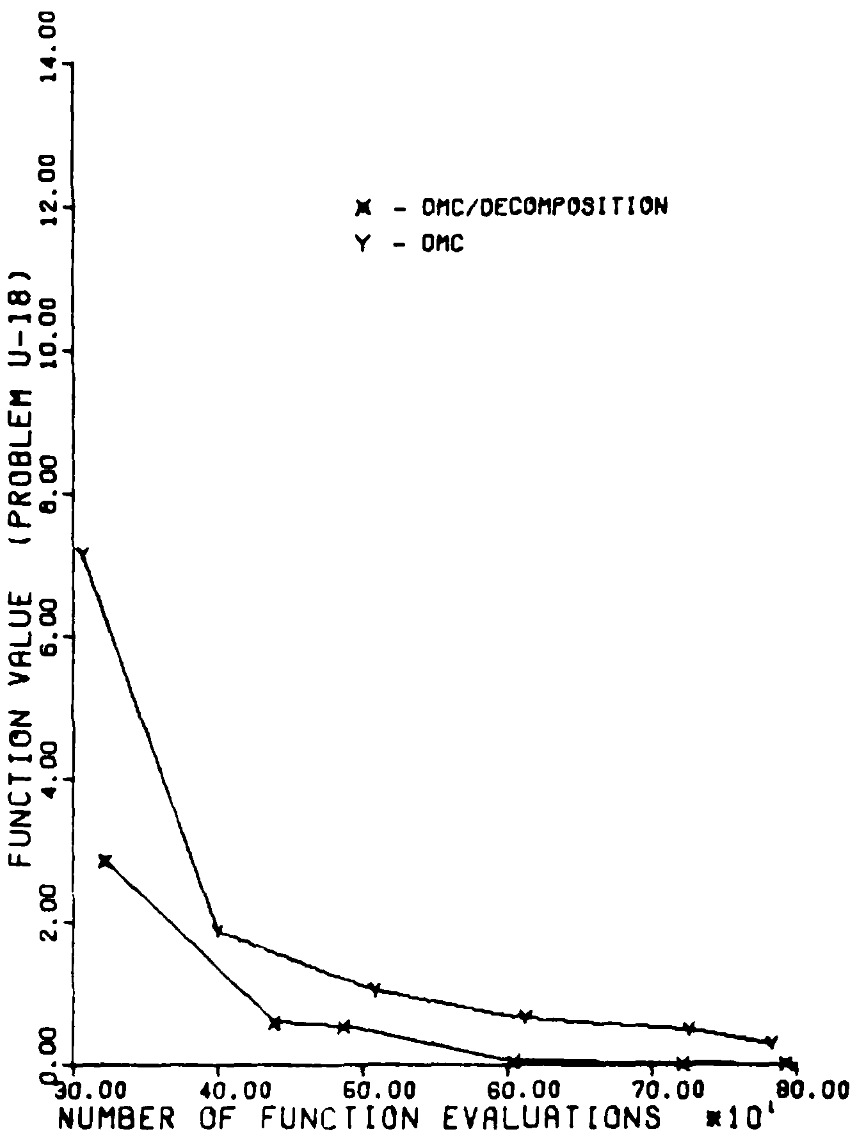


FIGURE 4-4. CHANGE IN FUNCTION VALUE VS NUMBER OF FUNCTION EVALUATIONS PROBLEM U-18

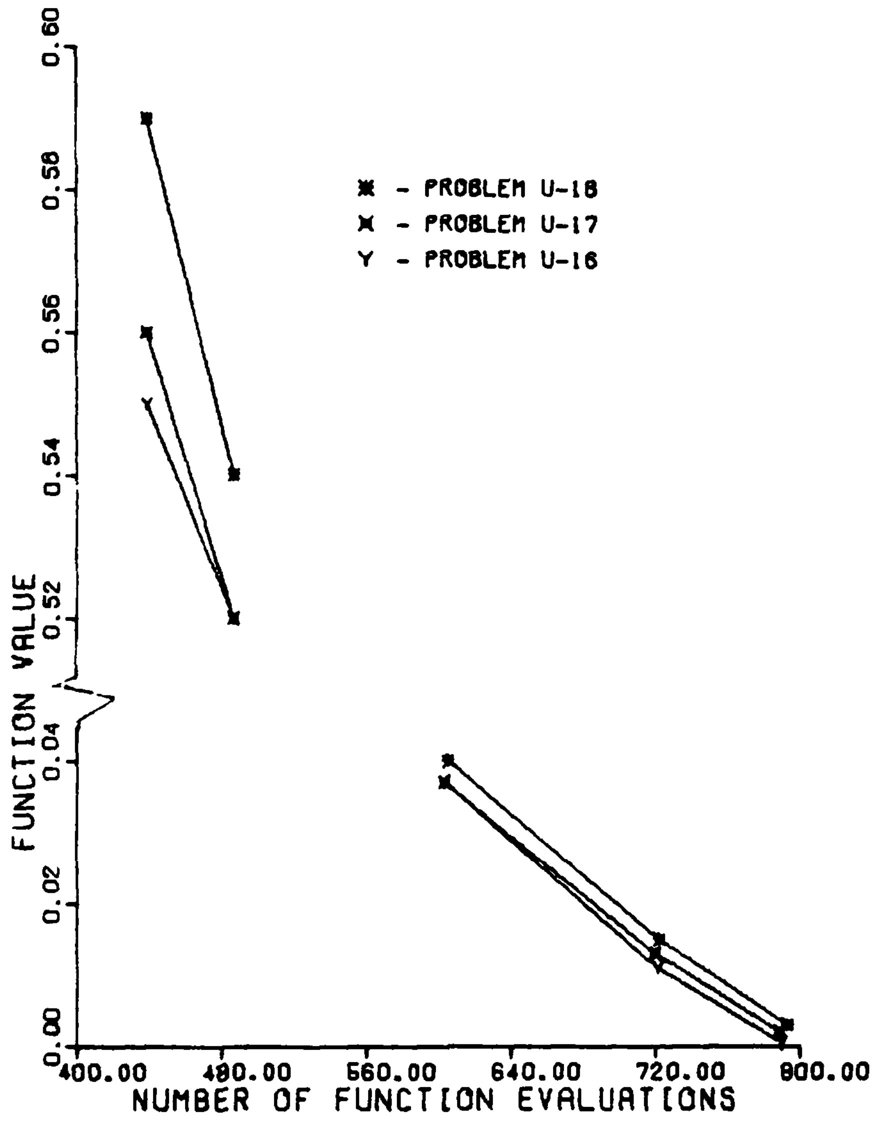


FIGURE 4-5, CHANGE IN FUNCTION VALUE VS NUMBER OF FUNCTION EVALUATIONS

Of course, problems U-16 and U-17 were designed so that they could be efficiently solved by the decomposition technique. Problem U-6C, on the other hand, is a widely known unconstrained test problem. Study of the objective function for this problem reveals that the two fourth power terms, which are proportional to the differences between the variable pair X(2),X(3) and the pair X(1),X(4), are dominant for the initial point. As the above pairs of variables come close together, and in particular as all the variables become small, the second power terms dominate. In these second power terms the pairs X(1),X(2) and X(3),X(4) interact within pairs but not between pairs. This problem was solved by using the decomposition procedure and the above observations to guide the selection of the subspaces to be searched. The results are summarized in the plot in Figure 4-6. Once again the decomposition method is more efficient than the DMC algorithm alone.

The examples above clearly illustrate that for problems with sets of variables with little or no interaction between sets of variables the decomposition approach is more efficient than the DMC algorithm. Also illustrated is the fact that the efficiency of the decomposition algorithm is highest when the interaction between groups of variables is lowest.

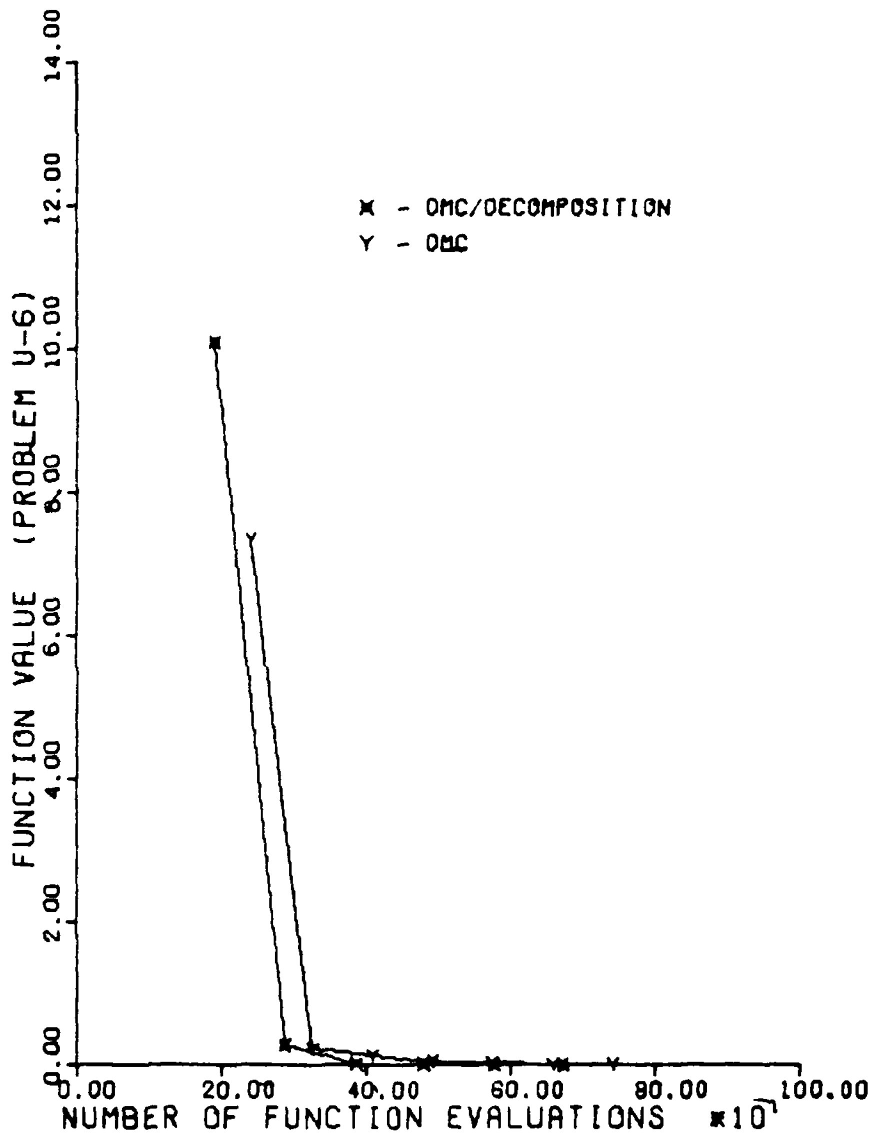


FIGURE 4-6. CHANGE IN FUNCTION VALUE VS NUMBER OF FUNCTION EVALUATIONS PROBLEM U-6C

4.6.3 A Grid Algorithm

The grid algorithm described in Chapter 3 was tried on problem C-4A. The initial grid had increments of .1 in each coordinate and subsequent grids with increments of .01 and .001 were used. After convergence of the DMC algorithm for a given grid, the next smaller increments were selected, the non-random complex generated using the previous solution as the base point, and the DMC algorithm restarted. A solution very close to that obtained by the DMC algorithm alone with objective function value of -5.267 was obtained after 941 objective function and 1174 constraint evaluations. This is about half the number of function and constraint evaluations used by the DMC algorithm alone.

On problem C-7A the reduction in number of function and constraint evaluations was less. A solution with objective function value -3.295 was obtained by the grid algorithm after 373 objective function and 481 constraint evaluations. This represents a reduction in objective function and constraint evaluations of about 10 percent.

4.6.4 Acceleration by Trajectory Analysis

The linear and quadratic trajectory analysis methods were described in Chapter 3. The results of applying these techniques, reported below, were gathered by using an acceleration search over the trajectory after each 10 iterations of the DMC algorithm. In each case where the search along the trajectory located a solution better than the current best solution, the current best vertex in the complex was replaced by the new point.

The progress of the DMC algorithm, as well as the DMC algorithm with trajectory analysis, on problem C-1 is illustrated in Figure 4-7. The three algorithm variations arrive at three different local optima. The best solution was found by the algorithm variant using quadratic trajectory analysis and the worst was found by the algorithm using linear trajectory analysis.

The results of using the trajectory based acceleration on problem U-4D are illustrated in Figure 4-8. On this problem all three algorithm variations arrive at the same solution but the algorithm variation using quadratic trajectories terminated after 907 function evaluations. The DMC algorithm required 1137 function evaluations and the variation using linear trajectories required 1329. That is, using quadratic trajectory analysis required 20 percent fewer and using linear trajectory analysis required 17 percent more function evaluations than did the DMC algorithm alone. Thus, the quadratic trajectory acceleration can result in locating the optimal solution with fewer function evaluations. For problems with multiple local optima, the

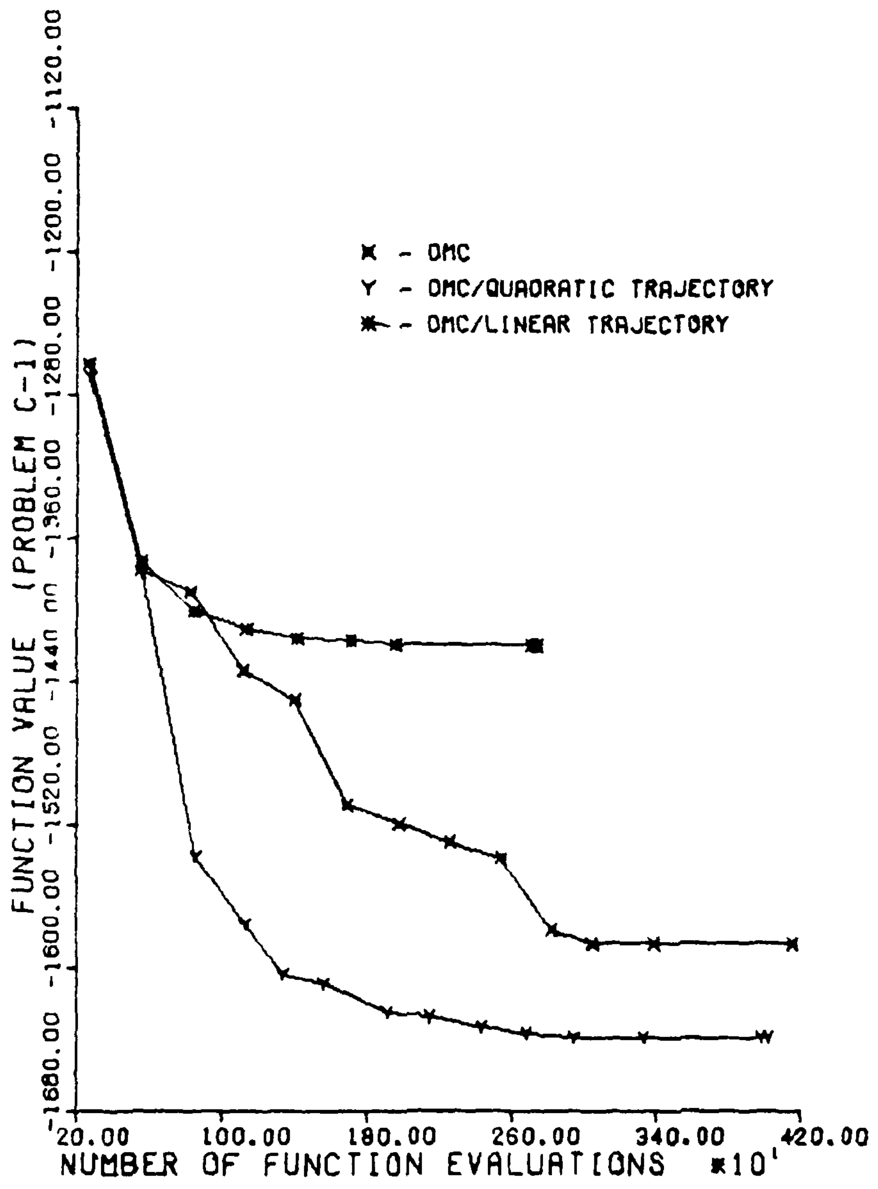


FIGURE 4-7. CHANGE IN FUNCTION VALUE VS NUMBER OF FUNCTION EVALUATIONS PROBLEM C-1

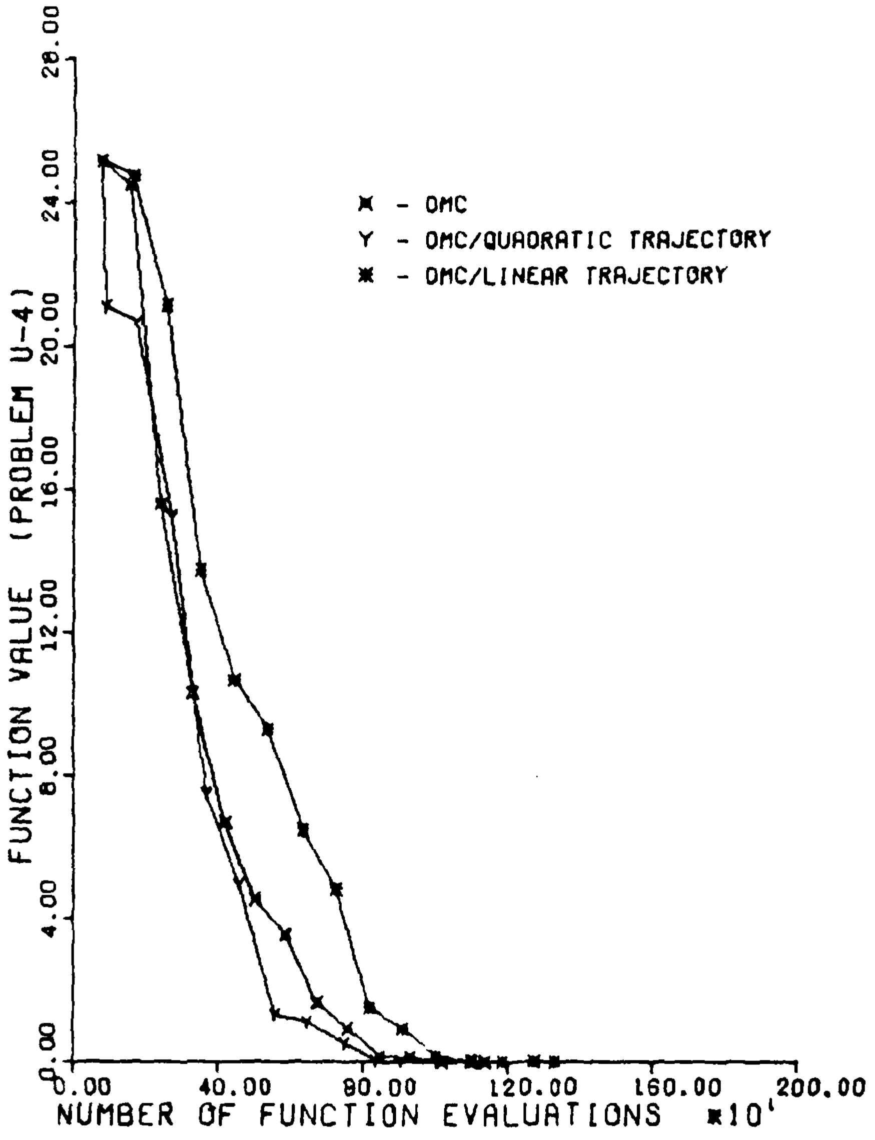


FIGURE 4-8. CHANGE IN FUNCTION VALUE VS NUMBER OF FUNCTION EVALUATIONS PROBLEM U-40

technique can divert the search so that different solutions may be located from the same initial complex. The fact that linear trajectories were less successful than the quadratic trajectories can be attributed to the tendency of the linear trajectory to introduce linear dependencies into the complex.

4.6.5 Conclusions

The above sampling of composite algorithm options has illustrated some successful applications. The ability of the effective objective function formulation to allow search from infeasible starting points was used to exploit the fact that useful solutions to discrete problems are sometimes found near the continuous optimum. A successful decomposition algorithm was easy to implement simply by initializing a starting complex to lie in a subspace. The grid algorithm, based on the discrete search algorithm, was demonstrated. Finally the quadratic acceleration scheme was shown to be a useful addition to the DMC algorithm. Some further ideas are discussed in the next chapter.

5 CONCLUSION

5.1 Summary

In this research a composite algorithm applicable to mixed integer constrained nonlinear problems has been developed. Throughout this research, the goal of solving engineering design problems has been used to guide the design of the algorithm. The composite algorithm is implemented as an interactive computer program that allows the designer to be involved in the optimization search. The major component of the composite algorithm is a version of the complex algorithm that incorporates modifications previously proposed, but not previously combined, as well as new modifications. The new modifications include the incorporation of a unidimensional search component, a new method for handling constraints based on an effective objective function formulation of the problem, and new termination criteria for the algorithm. Additional algorithmic elements incorporated into the composite algorithm include a new acceleration strategy based on trajectory analysis, a new decomposition approach, and a sequential grid reduction algorithm.

The modified complex algorithm has been tested on a variety of problems primarily selected to represent engineering design applications. Results of some options of the composite algorithm are reported by example. The results indicate that the modified complex algorithm is a useful method for solving discrete, constrained, nonlinear optimization problems and is more efficient than penalty function extensions to discrete unconstrained algorithms. Although the algorithm can successfully solve mixed integer problems as well, it was shown that the use of discrete approximation (treating variables as discrete with small stepsize) was superior to treating the continuous variables explicitly as continuous.

Some auxiliary techniques for the composite algorithm were useful adjuncts to the complex algorithm. In particular, the quadratic trajectory acceleration strategy was demonstrated to require fewer function evaluations than the DMC algorithm without acceleration. In addition, the acceleration proved useful in redirecting the path of the search so that, for problems with a number of local optima, different solutions could be located. The decomposition approach proposed is applicable to problems with a specific structure. These problems can be solved more quickly by successivly searching subspaces defined by groups of interacting variables than by searching over all variables simultaneously. The grid algorithm results in solving some problems using fewer objective function and constraint evaluations than used by the DMC algorithm alone. Finally, the effective objective function formulation for incorporating problem constraints allows infeasible starting points to be used. This is particularly useful when a feasible starting point is not readily available, or for searching for a discrete solution in a specific vicinity such as near the continuous optimum.

5.2 Suggestions for Further Research

In this research some new concepts for a search algorithm have been implemented and tested. The suggestions for additional research discussed below fall into three areas: (1) ideas for improving efficiency of the implementation of the algorithm, (2) suggestions for more general applicability of concepts developed in this research, and (3) some additional research ideas.

The alternative direction regeneration scheme for the complex algorithm was suggested by Beveridge and Schechter [4] and is an important part of the algorithm. Although efficacious it sometimes requires a large number of function (and for constrained problems, constraint) evaluations. On certain problems a large proportion of the total number of evaluations are due to this regeneration scheme. Two approaches are suggested for improving efficiency. Either an alternate regeneration method could be substituted or criteria could be developed to regulate the use of the existing method.

The discussion in Chapter 3 of the effective objective function formulation for handling constraints makes note of the fact that some efficiency can be gained by evaluating the constraints sequentially. Even greater efficiency could result from reordering constraint evaluations. The earlier the violated constraint is evaluated the greater the savings in computation. The simplest way to implement this is to request that the designer program the constraints most likely to be violated first. However, the designer may not know which constraints these will be. A second approach is to modify the program to store a simple history of which constraints are most often violated. Based on this history the order of evaluation of the constraints could be set dynamically by the program.

The decomposition approach developed in this research could use any search algorithm for search over the subspace. In addition, the technique may be applicable to a wide variety of problems because, at certain stages of solution, the required conditions for lack of strong interaction between subsets of variables may be temporarily satisfied. What is needed is a method to automatically select the subspace to be searched, letting groupings of the variables into subspaces change as the search progresses.

The trajectory analysis acceleration scheme developed in this research may also be applied to other search algorithms. Any sequence of solutions that show a trend in the objective function can be used for potential acceleration by extrapolation.

The aim of the quadratic acceleration scheme is to identify and search along a valley of the objective function. The current algorithm limits the total number of objective function evaluations on any one unidimensional search, thus, the accuracy in locating the lowest point in the valley is limited. It may be that, for those unidimensional searches that are used to locate points that are later used in defining a quadratic trajectory, additional accuracy in locating a minimum is justified. The additional accuracy in locating the minimum on the unidimensional searches should yield a more accurate identification of the valley and hence more progress may be made in searching along the trajectory. The trade-off between the accuracy on the unidimensional searches (and simultaniously the number of objective function evaluations) and the success of the quadratic trajectory extrapolation should be investigated.

Another trade off to be evaluated is the size of the initial complex and the robustness of the modified complex algorithm. The nonrandom starting complex used in this research was made as large as possible by setting variables to the bounding values (see Chapter 3). Of course the variables could be set any percentage of the distance from the initial point to the boundry. The large initial complex is expected to result in the most thorough search of the feasible region. A smaller initial complex might be expected to shorten the search, that is, convergence criteria should be satisfied after fewer objective function evaluations.

The following modification to the complex algorithm could prevent the distortion of the complex which occurs when a unidimensional search locates a point that is remote from the other points in the complex. Rather than proceeding with this distorted complex, it might be advantageous to create a new complex about this point. This could be done either by translating the existing complex or by using the complex initialization scheme.

Finally, it can be noted that as the complex algorithm proceeds, redundancies can occur in the complex vertexes. That is, the same, or nearly the same, point may be present as more than one vertex. This can be avoided by periodically generating a new complex with the initialization scheme or by using a solution found by the quadratic trajectory acceleration scheme to replace a redundant vertex.

In summary, this research has developed and implemented a new optimization algorithm which is particularly suited for engineering design problems. The algorithm is implemented in the form of an interactive, composite algorithm. Elements of the algorithm employ straightforward techniques to enforce discreteness of variables and problem constraints. The algorithm has been tested using problems selected to represent engineering design applications, and the success of the algorithm on these problems is most promising.

REFERENCES

[1] Avriel, M., Nonlinear Programming Analysis and Methods, Prentice-Hall Inc., 1976

[2] Bartle, R. G., The Elements of Real Analysis, John Wiley and Sons, 1964

[3] Beightler, C. S. and Philips, D. T., Applied Geometric Programming, John Wiley and Sons, 1976

[4] Beveridge, G. S. and Schechter, R. S., Optimization: Theory and Practice, McGraw-Hill, 1970

[5] Box, M. J., "A New Method of Constrained Optimization and a Comparison with Other Methods", Computer Journal, Vol. 8, No. 1, Apr. 1965, pp. 42-52

[6] Box, M. J., "A Comparison of Several Current Optimization Methods, and the Use of Transformations in Constrained Problems", Computer Journal, Vol. 9, No. 1, May 1966, pp. 67-77

[7] Buffa, E. S., "Empirical Tests of Constrained Nonlinear Optimization Algorithms", Decision Sciences, Vol. 8, No. 2, Apr. 1977, pp. 445-464

[8] Carroll,C.W., "The Created Response Surface Technique for Optimizing Nonlinear Restrained Systems", Operations Research, Vol. 9, No. 2, Mar-Apr. 1961, pp. 168-184

[9] Cella, A. and Soosaar, K., "Discrete Variables in Structural Optimization", in Optimum Structural Design, ed. Gallagher, R. H. and Zienkiewicz, O. C., John Wiley and Sons, 1973

[10] Chanaratna, V., "Discrete Structural Optimization", Ph. D. Dissertation, Department of Civil Engineering, University of Illinois, 1978

[11] Colville, A. R., "A Comparative Study of Nonlinear Programming Codes", IBM N.Y. Science Center Report 320-2949, Jun. 1968

[12] Courant, R., "Variational Methods for the Solution of Problems of Equilibrium and Vibrations", Bulletin of the American Mathematical Society, Vol. 49, Jan.-Dec. 1943, pp. 1-23

[13] Davies, D. and Swann, W. H., "Review of Constrained Optimization", in Optimization, ed. Fletcher, R., Academic Press, 1969, pp. 247-258

[14] de Silva, B. M. E., "The Application of Nonlinear Programming to the Automated Minimum Weight Design of Rotating Disks", in Optimization, ed. Fletcher, R., Academic Press, 1969, pp. 115-150

[15] Eason, E. D. and Fenton, R. G., "A Comparison of Numerical Optimization Methods for Engineering Design", Transactions of the ASME, Journal of Engineering for Industry, Vol. 96, Ser. B, No. 1, Feb. 1974, pp. 196-200

[16] Fiacco, A. V. and McCormick, G. P., Nonlinear Programming: Sequential Unconstrained Minimization Techniques, John Wiley and Sons, 1967

[17] Fiacco, A. V. and McCormick, G. P., "The Sequential Unconstrained Minimization technique for Nonlinear Programming, A Primal-Dual Method", Management Science, Vol. 10, No. 2, Jan. 1964, pp. 360-365

[18] Fletcher, R., "A Review of Methods for Unconstrained Optimization", in Optimization, ed. Fletcher, R., Academic Press, 1969

[19] Fletcher, R., "Function Minimization without Evaluating Derivatives - a Review", Computer Journal, Vol. 8, No. 1, Apr. 1965, pp. 33-41

[20] Fletcher, R. and Powell, M. J. D., "A Rapidly Convergent Descent Method for Minimization", Computer Journal, Vol. 6, No. 2, Jul. 1963, pp. 163-168

[21] Fox, R. L., Optimization Methods for Engineering Design, Addison-Wesley, 1971

[22] Gallagher, R. H., "Fully Stressed Design", in Optimum Structural Design, ed. Gallagher, R. H. and Zienkiewicz, O.C., John Wiley and Sons, 1973, pp. 19-32

[23] Gallagher, R. H., "Terminology and Concepts", in Optimum Structural Design, ed. Gallagher, R. H. and Zienkiewicz, O.C., John Wiley and Sons, 1973, pp. 7-17

[24] Gisvold, K. M. and Moe, J., "A Method for Nonlinear Mixed-Integer Programming and its Application to Design Problems", Transactions of the ASME, Journal of Engineering for Industry, Vol. 94, Ser. B, No. 2, May 1972, pp. 353-364

[25] Glankwahmdee, A., "Unconstrained Nonlinear Discrete Search", Ph. D. Dissertation, Department of Mechanical and Industrial Engineering, University of Illinois, 1976

[26] Glankwahmdee, A., Liebman, J. S. and Hogg, G. L., "Unconstrained Discrete Nonlinear Programming", Engineering Optimization, Vol. 4, No. 2, 1979, pp. 95-107

[27] Glover, F., and Sommer, D., "Pitfalls of Rounding in Discrete Management Decision Problems", Decision Science, Vol. 22, No. 4, Dec. 1975, pp. 455-460

[28] Grey, D. S., "Boundry Conditions in Optimization Problems", in Recent Advances in Optimization Techniques, ed. Lavi, A. and Vogl, T. P., John Wiley and Sons, 1965

[29] Guin, J. A., "Modification of the Complex Method of Constrained Optimization", Computer Journal, Vol. 10, No. 4, Feb. 1968, pp. 416-417

[30] Hati, S. K. and Rao, S. S., "Determination of Optimum Machining Conditions - Deterministic and Probilistic Approaches", Transactions of the ASME, Journal of Engineering for Industry, Vol. 98, Ser. B, No. 1, Feb. 1976, pp. 354-359

[31] Heinin, C., "Computational Techniques for Optimizing Systems with Standby Redundancy", Naval Research Logistics Quarterly, Vol. 19, No. 2, Jun. 1972, pp. 293-308

[32] Himmelblau, D. M., Applied Nonlinear Programming, McGraw-Hill, 1972

[33] Himmelblau, D. M., "A Uniform Evaluation of Unconstrained Optimization Techniques:, in Numerical Methods for Non-Linear Optimization, ed. Lootsma, F. A., Academic Press, 1972, pp. 69-97

[34] Hooke, R. and Jeeves, T. A., "Direct Search Solution of Numerical and Statistical Problems", Journal of the Association for Computing Machinery, Vol. 8, No. 2, Apr. 1961, pp. 212-229

[35] Ibaraki, T., Ohashi, T. and Mine, H., "A Heuristic Algorithm for Mixed-Integer Programming Problems", in Mathematical Programming Study 2, North-Holland Publishing Co., 1974, pp. 115-136

[36] Khachaturian, N. and Horowitz, B., "Properties of Optimal Structures", Proceedings of the Symposium on Applications of Computer Methods in Engineering, ed. Wellford, L. C. Jr., University of Southern California, Vol. 1, Aug. 1977, pp. 533-542

[37] Kuester, J. L. and Mize, J. H., Optimization Techniques with Fortran, McGraw-Hill, 1970

[38] Land, A. N., and Doig, A. G., "An Automatic Method of Solving Discrete Programing Preblems", Econometrics, Vol 28, No. 3, Jul. 1960, pp. 497-520

[39] Lasdon, L. S., Fox, R. L. and Ratner, M. W. , "An Efficient One-Dimensional Search Proceedure for Barrier Functions", Mathematical Programming, Vol. 4, No. 3, Jun. 1973, pp. 279-296

[40] Liebman, J. S., Chanaratna, V., and Khachaturian, N., "Discrete Optimization in Structural Design", Proceedings of the Symposium on Applications of Computer Methods in Engineering, ed. Wellford, L. C. Jr., University of Southern California, Vol. 1, Aug. 1977, pp. 553-562

[41] Luus, R., "Optimization of System Reliability by a New Nonlinear Integer Programing Procedure", IEEE Transactions on Reliability, Vol. R74, No. 1, Apr. 1975, pp. 14-16

[42] Mischke, C. R., An Introduction to Computer-Aided Design, Prentice-Hall Inc., 1968

[43] Mitchel, R. A. and Kaplan, J. L., "Nonlinear Constrained Optimization by a Nonrandom Complex Method", Journal of Research of the National Bureau of Standards, Engineering and Instrumentation, Vol. 72C, No. 4, Oct.-Dec. 1978, pp. 249-258

[44] Moe, J., "Penalty-Function Methods", in Optimum Structural Design, ed. Gallagher, R. H. and Zienkeiwicz, O. C., John Wiley and Sons, 1973, pp. 143-177

[45] Murray, W., "An Algorithm for Constrained Minimization", in Optimization, ed. Fletcher, R., Academic Press, 1969, pp. 247-258

[46] Nelder, J. A. and Mead, R., "A Simplex Method for Function Minimization", Computer Journal, Vol. 7, No. 4, Jan. 1965, pp. 308-313

[47] Pappas, M., "Use of Direct Search in Automated Optimal Design", Transactions of the ASME, Journal of Engineering for Industry, Vol. 94, Ser. B, No. 2, May 1972, pp. 395-401

[48] Pappas, M. and Allentuch, A., "Mathematical Programming Proceedures for Mixed Discrete-Continuous Design Problems, Transactions of the ASME, Journal of Engineering for Industry, Vol 96, Ser. B, No. 1, Feb. 1974, pp. 201-209

[49] Pappas, M. and Amba-Roa, C. L., "A Direct Search Algorithm for Automated Optimum Structural Design", AIAA Journal, Vol. 9, No. 3, Mar. 1971, pp. 387-393

[50] Pappas, M. and Moradi, J. Y., "An Improved Direct Search Mathematical Programming Algorithm", Transaction of the ASME, Journal of Engineering for Industry, Vol. 96, Ser. B, No. 4, Nov. 1975, pp. 1305-1310

[51] Parkinson, J. M. and Hutchinson, D., "A Consideration of Non-gradient Algorithms for the Unconstrained Optimization of Functions of High Dimensionality", in Numerical Methods for Non-linear Optimization, ed. Lootsma, F. A., Academic Press, 1972, pp. 69-97

[52] Parkinson, J. M. and Hutchinson, D., "An Investigation into Efficiency of Variants on the Simplex Method", in Numerical Methods for Non-linear Optimization, ed. Lootsma, F. A., Academic Press, 1972, pp. 115-135

[53] Paviani, D. A. and Himmelblau, D. M., "Constrained Nonlinear Optimization by Heuristic Programming", Operations Research, Vol. 17, 1969, pp. 872-882

[54] Powell, M. J. D., "An Efficient Method for Finding the Minimum of a Function of Several Variables Without Calculating Derivatives:, Computer Journal, Vol. 7, No. 2, Jul. 1964, pp. 155-162

[55] Ragsdell, K. M., "On Some Experiments Which Delimit the Utility of Nonlinear Programming Algorithms", Working Paper, Purdue University, 1978

[56] Ramamoorty, M. and Rao, P. J., "Comparative Study of Optimization Methods for the Design of Polyphase Reluctance Motors", Engineering Optimization, Vol. 3, No. 1, 1977, pp. 51-60

[57] Rao, S. S. and Kumar, A., "Optimization of Cold Rolling by Nonlinear Programming:, Transactions of the ASME, Journal of Engineering for Industry, Vol. 100., Ser. B, No. 2, May 1978, pp. 186-192

[58] Reiter, S. and Rice, D. B., "Discrete Optimizing Solution Proceedures for Linear and Nonlinear Integer Programming Problems", Management Science, Vol. 12, No. 11, Jul. 1966, pp. 829-850

[59] Rockafeller R. T., Convex Analysis, Princeton University Press, 1970

[60] Root, R. R. and Ragsdell, K. M., "A Survey of Optimization Methods Applied to the Design of Mechanisms", Transactions of the ASME, Journal of Engineering for Industry, Vol. 98, Ser. B, No. 3, Aug. 1976, pp. 1036-1041

[61] Rosenbrock, H. H. and Storey, C., Computational Techniques for Chemical Engineers, Pergamon Press, 1966

[62] Schmit, L. A., "Automated Design", International Science and Technology, No. 54, June 1966, pp. 63-78 and 115-117

[63] Schmit, L. A. Jr., Kicher, T. P. and Morrow, W. M., "Structural Synthesis Capability of Integrally Stiffened Waffle Plates", AIAA Journal, Vol. 1, No. 12, Dec. 1973, pp. 2820-2836

[64] Schuldt, S. B., Gabriele, G. A., Root, R. R., Sangren, E. and Ragsdell, K. M., "Application of a New Penalty Function Method to Design Optimization", Transactions of the ASME, Journal of Engineering for Industry, Vol. 99, Ser. B. No. 1, Feb. 1977, pp. 31-36

[65] Sheu, C. Y. and Prager, W., "Recent Developments in Optimal Structural Design", Applied Mechanics Review, Vol. 21, No. 10, Oct. 1968, pp. 985-992

[66] Simmons, L. M. and Pike, D. H., "A Mixed Integer Direct Search Technique for the Optimization of Constrained Non-linear Objective Functions", Working Paper, University of Tennessee

[67] Seireg, A., "A Survey of Optimization of Mechanical Design", Transactions of the ASME, Journal of Engineering for Industry, Vol. 94, Ser. B, No. 2, May 1972, pp. 495-499

[68] Shanno, D. F. and Weil, R. L., "Management Science: A View from Nonlinear Programming", Communications of the Association for Computing Machinery, Vol. 15, No. 7, Jul. 1972, pp. 542-549

[69] Smith, E. A. and Carpenter, W. C., "A Feasible Direction Method Based on Zoutendijk's Procedure P1", Engineering Optimization, Vol. 3, No. 2, Jan. 1978, pp. 109-112

[70] Spendley, W. and Hext, G. R., and Himsworth, R. R., "Sequential Applications of Simplex Designs in Optimization and Evolutionary Operation", Technometrics, Vol. 4, No. 4, Nov. 1962, pp. 441-461

[71] Stoecker, W. F., Design of Thermal Systems, McGraw-Hill, 1971

[72] Taha, H. A., Integer Programming Theory, Applications, and Computations, Academic Press, 1975

[73] Tillman, F. A., Ching-Lia, H. and Way, K., "Determining Component Reliability and Redundancy for Optimum System Reliability", IEEE Transactions on Reliability, Vol. R26, No. 3, Aug. 1977, pp. 162-165

[74] Tuthill, S., ME393 Project Report to Prof. C. O. Pederson, University of Illinois, 1978

[75] Wasiutynski, Z. and Brandt, A., "The Present State of Knowledge in the Field of Optimum Design of Structures", Applied Mechanics Reviews, Vol. 16, No. 5, May 1963, pp. 341-350

[76] Weisman, J. and Wood, C. F., "The Use of Optimal Search for Engineering Design", in Recent Advances in Optimization Techniques, ed. Lavi, A. and Vogl, T. P., John Wiley and Sons, 1965, pp. 219-228

[77] Wilde, D. J. and Beightler, C. S., Foundations of Optimization, Prentice-Hall Inc., 1967

[78] Zangwill, W. I., "Nonlinear Programing Via Penalty Functions", Management Science, Vol. 13, No. 5, Jan. 1967, pp. 344-358

APPENDIX 1.

This appendix contains the detailed numerical results that are summarized in Chapter 4. The source of each problem and a short description are given. Where the objective function (and for constrained problems, the constraint functions) is/are not simple expressions the reader is referred to Appendix 2 where the FORTRAN code for computing the objective and constraint functions is given. Finally, for each algorithm tried on a problem, the number of objective and constraint function evaluations, the objective function value and the solution vector obtained are given.

Problems labeled C-1 to C-21 are the constrained test problems. Problems labeled U-1 to U-15 are the unconstrained test problems.

PROBLEM: C-1

SOURCE: Beightler and Phillips [3], example 11.15, Alkylation process.

VARIABLES: 5

CONSTRAINTS: 14

MINIMIZE: see Appendix 2

SUBJECT TO: see Appendix 2

BOUNDS: $1. \leq X(1) \leq 2000.$

$1. \leq X(2) \leq 16000.$

$1. \leq X(3) \leq 120.$

$1. \leq X(4) \leq 5000.$

$90. \leq X(5) \leq 95.$

$1.2 \leq X(6) \leq 4.$

$145. \leq X(7) \leq 162.$

INCREMENTS: (.001 , .001 , .001 , .001 , .001 , .001 , .001)

STARTING POINT: (1745. , 12000. , 110. , 3048. , 92.8 , 3.6 , 145.)

ALGORITHM	NUMBER OF FUNCTION/ CONSTRAINT EVALUATIONS	RESULT
GRG	29/29	-942.207 = F(1744.9969 , 11999.9999 , 109.9815 , 3048.0032 , 93.1549 , 93.1549 , 3.6 , 145)
FLEX	3161/146566	-1594.666 = F(1761.791 , 13584.0 , 78.4221 , 3083.435 , 94.9969 , 1.6019 , 153.4129)
DMC	2151/2829	-1586.508 = F(1769.790 , 13695.600 , 78.781 , 3089.990 , 94.999 , 1.613 , 153.317)

PROBLEM: C-2

SOURCE: Fox [21], design of two bar truss.

VARIABLES: 2

CONSTRAINTS: 2

MINIMIZE: F = .6 * 3.14159 * X(2) / 4. * .1 * SQRT(900. + X(1)**2)

SUBJECT TO: see Appendix 2

*** VARIATION A ***

BOUNDS: $10. \leq X(1) \leq 35.$

$4. \leq X(2) \leq 12.$

INCREMENTS: (.001 , .001)

STARTING POINT: (30. , 10.)

ALGORITHM	NUMBER OF FUNCTION/ CONSTRAINT EVALUATIONS	RESULT
GRG	57/57	12.813 = F(20.2369, 7.5134)
FLEX	820/3893	12.812 = F(20.2369, 7.5134)
DMC	141/200	12.813 = F(20.235 , 7.514)

*** VARIATION B ***

BOUNDS: as in variation A

INCREMENTS: (1. , 1.)

STARTING POINT: as in variation A

ALGORITHM	NUMBER OF FUNCTION/ CONSTRAINT EVALUATIONS	RESULT
GRG/R/N1	63/63	13.387 = F(19. , 8.)
G-2/P	212/212	13.387 = F(19. , 8.)
DMC	28/38	13.387 = F(19. , 8.)

PROBLEM: C-3

SOURCE: Eason and Fenton [15], problem 6, Journal Bearing Design.

VARIABLES: 2

CONSTRAINTS: 1

MINIMIZE: F=(.44 * X(1)**3/X(2)**2 + 10. * X(1) + .592 * X(1)/X(2)**3)/10.

SUBJECT TO: 8.63 * X(2)**3 / X(1) -1. $\leq$ 0.

*** VARIATION A ***

BOUNDS: .1 $\leq$ X(i) $\leq$ 5. , i=1,2

INCREMENTS : (.001, .001)

STARTING POINT: (2.5 , 2.5)

ALGORITHM	NUMBER OF FUNCTION/ CONSTRAINT EVALUATIONS	RESULT
GRG	63/63	1.621 = F(1.2873 , .5305)
FLEX	321/2252	12.813 = F(20.237 , 7.513)
DMC	338/573	1.621 = F(1.291 , .531)

*** VARIATION B ***

BOUNDS: as in variation A

INCREMENTS: as in variation A

STARTING POINT: (3. , .7)

ALGORITHM	NUMBER OF FUNCTION/ CONSTRAINT EVALUATIONS	RESULT
GRG	94/94	1.621 = F(1.2873 , .5305)
FLEX	210/933	1.621 = F(1.2886 , .5307)
DMC	195/294	1.621 = F(1.298 , .532)

*** VARIATION C ***

BOUNDS: as in variation A

INCREMENTS: (.1 , .1)

STARTING POINT: (2.5 , 2.5)

ALGORITHM	NUMBER OF FUNCTION/ CONSTRAINT EVALUATIONS	RESULT
GRG/R/N1	99/99	1.672 = F(1.125 , .5)
G-2/P		NO FEASIBLE SOLUTION
DMC	87/178	1.672 = F(1.125 , .5)

*** VARIATION D ***

BOUNDS: as in variations A

INCREMENTS: (.1 , .1)

STARTING POINT: (3. , .7)

ALGORITHM	NUMBER OF FUNCTION/ CONSTRAINT EVALUATIONS	RESULT
GRG/R/N1	68/68	1.672 = F(1.125 , .5)
G-2/P	309/309	1.736 = F(1.25 , .5)
DMC	51/93	1.672 = F(1.125 , .5)

PROBLEM: C-4

SOURCE: Box [5], problem A. Himmelblau [32], problem 13.

VARIABLES: 5

CONSTRAINTS: 6

MINIMIZE: F = (C0 + C1 * X(1) + C2 * X(1) * X(2) + C3 * X(1) * X(3) +
C4 * X(1) * X(4) + C5 * X(1) * X(5)) * (-.000001)

SUBJECT TO: 0. $\leq$ C6*X(1) + C7*X(1)*X(2) + C8*X(1)*X(3) +C9*X(1) * X(4) +
C10 * X(1) *X(5) $\leq$ 294000.

AND: 0. $\leq$ C11*X(1) + C12*X(1)*X(2) + C13*X(1)*X(3) + C14*X(1) *
X(4) + C15 * X(1) * X(5) $\leq$ 294000.

AND: 0. $\leq$ C16*X(1) + C17*X(1)*X(2) + C18*X(1)*X(3) + C19*X(1) *
X(4) + C20 * X(1) * X(5) $\leq$ 277200.

*** VARIATION A ***

BOUNDS: 0. $\leq$ X(1) $\leq$ 5.

1.2 $\leq$ X(2) $\leq$ 2.4

20. $\leq$ X(3) $\leq$ 60.

9.0 $\leq$ X(4) $\leq$ 9.3

6.5 $\leq$ X(5) $\leq$ 7.

INCREMENTS: (.001 , .001 , .001 , .001 , .001)

STARTING POINT: (2.52 , 2. , 37.5 , 9.25 , 6.8)

ALGORITHM	NUMBER OF FUNCTION/ CONSTRAINT EVALUATIONS	RESULT
GRG	60/60	-5.208 = F(4.5374 , 2.4 , 60. , 9.3 , 7)
FLEX	1720/50991	-5.262 = F(4.5584 , 2.3383 , 59.999 , 9.2997 , 7.)
DMC	2180/2593	-5.276 = F(4.559 , 2.379 , 59.894 , 9.299 , 6.996)

*** VARIATION B ***

BOUNDS: as in variation A

INCREMENTS: (1. , 1. , 1. , 1. , 1.)

STARTING POINT: as in variation A

ALGORITHM	NUMBER OF FUNCTION/ CONSTRAINT EVALUATIONS	RESULT
GRG/R/N1	66/66	-4.806 = F(5. , 2. , 59. , 9. , 7.)
G-2/P	38/38	-4.704 = F(5. , 2. , 60. , 9. , 7.)
DMC	69/69	-4.837 = F(5. , 2. , 20. , 9. , 7.)

*** VARIATION C ***

BOUNDS: as in variations A

INCREMENTS: (.25 , .25 , .25 , .25 , .25)

STARTING POINT: as in variation A

ALGORITHM	NUMBER OF FUNCTION/ CONSTRAINT EVALUATIONS	RESULT
GRG/R/N1	66/66	-5.028 = F(4.5 , 2.25 , 60. , 9.25 , 7.)
G-2/P	153/153	-5.054 = F(4.5 , 2.25 , 20. , 9.25 , 7.)
DMC	91/102	-5.153 = F(4.75 , 2. , 28.75 , 9.25 , 7.)

PROBLEM: C-5

SOURCE: Box [5], problem B.

VARIABLES: 2

CONSTRAINTS: 3

MINIMIZE: F = -((9. - (X(1) - 3)**2) * X(2)**3 / (27. * SQRT(3.)))

SUBJECT TO: 0. $\leq$ X(1) + SQRT(3.) * X(2) $\leq$ 6.

AND: X(2) - X(1) / SQRT(3.) $\leq$ 0.

BOUNDS: 0. $\leq$ X(i) $\leq$ 100. , i = 1,2

INCREMENTS: (.001 , .001)

STARTING POINT: (1. , .5)

ALGORITHM	NUMBER OF FUNCTION/ CONSTRAINT EVALUATIONS	RESULT
GRG	19/19	-1. = F(3. , 1.7321)
FLEX	354/2740	-1. = F(3. , 1.7320)
DMC	142/212	-1. = F(3. , 1.732)

PROBLEM: C-6

SOURCE: Eason and Fenton [15], problem 7, Flywheel design.

VARIABLES: 3

CONSTRAINTS: 2

MINIMIZE: F = (-.0201 * X(1)**4 * X(2) * X(3)**2) / 1.E7

SUBJECT TO: X(1)**2 * X(2) - 675. $\leq$ 0.

AND: (X(1) * X(3))**2 / 1.E7 - .419 $\leq$ 0.

*** VARIATION A ***

BOUNDS: 0. $\leq$ X(1) $\leq$ 36.

0. $\leq$ X(2) $\leq$ 5.

0. $\leq$ X)3) $\leq$ 125.

INCREMENTS: (.001 , .001 , .001)

STARTING POINT: (22.3 , .5 , 125.)

ALGORITHM	NUMBER OF FUNCTION/ CONSTRAINT EVALUATIONS	RESULT
GRG	71/71	-5.685 = F(16.3756 , 2.5172 , 125.)
FLEX	630/7808	-5.682 = F(17.187 , 2.2842 , 119.102)
DMC	530/704	-5.6846= F(31.669, .673 , 64.635)

*** VARIATION B ***

BOUNDS: as in variation A

INCREMENTS: (1. , 1. , 1.)

STARTING POINT: as in variation A

ALGORITHM	NUMBER OF FUNCTION/ CONSTRAINT EVALUATIONS	RESULT
GRG/R/N1	75/75	-4.770 = F(15. , 3. , 125)
G-2/P		NO FEASIBLE SOLUTION
DMC	79/114	-5.151 = F(25. , 1. , 81.)

*** VARIATION C ***

BOUNDS: as in variation A

INCREMENTS: (.5 , .5 , .5)

STARTING POINT: as in variation A

ALGORITHM	NUMBER OF FUNCTION/ CONSTRAINT EVALUATIONS	RESULT
GRG/R/N1	75/75	-5.146 = F(16. , 2.5 , 125)
G-2/P		NO FEASIBLE SOLUTION
DMC	109/148	-5.294 = F(36. , .5 , 56.)

*** VARIATION D ***

BOUNDS: as in variation A

INCREMENTS: (.25 , .25 , .25)

STARTING POINT: as in variation A

ALGORITHM	NUMBER OF FUNCTION/ CONSTRAINT EVALUATIONS	RESULT
GRG/R/N1	75/75	-5.475 = F(16.25 , 2.5 , 125.)
G-2/P		NO FEASIBLE SOLUTION
DMC	169/220	-5.436 = F(36. , .5 , 56.75)

PROBLEM: C-7

SOURCE: Eason and Fenton [15], problem 2, post office parcel.

VARIABLES: 3

CONSTRAINTS: 2

MINIMIZE: -X(1) * X(2) * X(3) * .001

SUBJECT TO: $0. \le X(1) + 2. * (X(2) + X(3)) \le 72.$

*** VARIATION A ***

BOUNDS: $0. \le X(1) \le 20.$

$0. \le X(2) \le 11.$

$0. \le X(3) \le 42.$

INCREMENTS: (.001 , .001 , .001)

STARTING POINT: (10. , 10. , 10.)

ALGORITHM	NUMBER OF FUNCTION/ CONSTRAINT EVALUATIONS	RESULT
GRG	25/25	-3.3 = F(20. , 11. , 15.)
FLEX	373/3796	-3.3 = F(19.9990 , 10.9977 , 15.0028)
DMC	417/512	-3.299 = F(19.982 , 11.0 , 15.009)

*** VARIATION B ***

BOUNDS: as in variation A

INCREMENTS: (1. , 1. , 1.)

STARTING POINT: as in variation A

ALGORITHM	NUMBER OF FUNCTION/ CONSTRAINT EVALUATIONS	RESULT
GRG/R/N1	29/29	-3.3 = F(20. , 11. , 15.)
G-2/P	126/126	-3.075 = F(20. , 11. , 14.)
DMC	91/117	-3.2 = F(20. , 10. , 16.)

PROBLEM: C-8

SOURCE: Himmelblau [32], problem 4, Chemical equalibrum.

VARIABLES: 7

CONSTRAINTS: 6

MINIMIZE: see Appendix 2

SUBJECT TO: see Appendix 2

BOUNDS: $0. \le X(1) \le 2.$

$0. \le X(5) \le .5$

$0. \le X(i) \le 1.$, $i = 2,3,4,6,7$

INCREMENTS: (.001 , .001 , .001 , .001 , .001 , .001 , .001)

STARTING POINT: (.25 , .25 , .25 , .25 , .25 , .25 , .25)

ALGORITHM	NUMBER OF FUNCTION/ CONSTRAINT EVALUATIONS	RESULT
GRG	89/89	-46.566 = F(.2214 , .3429 , .3159 , 0. , .5 , 0. , .2231)
FLEX	272/6004	-47.623 = F(.7170 , .1761 , .6976 , .0038 , .4878 , .0205 , .0009)
DMC	1815/1944	-47.761 = F(.042 , .142 , .788 , .001 , .486 , .001 , .018)

PROBLEM: C-9

SOURCE: Eason and Fenton [15], problem 1. Himmelblau [32], problem 10. Colville [11], problem 1.

VARIABLES: 5

CONSTRAINTS: 10

MINIMIZE: see Appendix 2

SUBJECT TO: see Appendix 2

*** VARIATION A ***

BOUNDS: $0. \leq X(i) \leq 100.$, i = 1,...,5

INCREMENTS: (.001 , .001 , .001 , .001 , .001)

STARTING POINT: (0. , 0. , 0. , 0. , 1.)

ALGORITHM	NUMBER OF FUNCTION/ CONSTRAINT EVALUATIONS	RESULT
GRG	113/113	-32.349 = F(.3 , .3335 , .4 , .4283 , .2240)
FLEX	395/9653	-32.348 = F(.3 , .3329 , .4 , .4272 , .2253)
DMC	714/948	-26.891 = F(0. , .389 , .250 , .721 , .257)

*** VARIATION B ***

BOUNDS: as in A

INCREMENTS: as in A

STARTING POINT: (1. , 1. , 1. , 1. , 1.)

ALGORITHM	NUMBER OF FUNCTION/ CONSTRAINT EVALUATIONS	RESULT
GRG	95/95	-32.349 = F(.3 , .3335 , .4 , .4283 , .2240)
FLEX	812/17056	-32.349 = F(.3 , .3335 , .4 , .4283 , .2240)
DMC	703/1076	-32.131 = F(.283, .340, .392 , .449 , .216)

PROBLEM: C-10

SOURCE: Eason and Fenton [15], problem 3. Himmelbleu [32], problem 11., Colville [11], problem 3

VARIABLES: 5

CONSTRAINTS: 6

MINIMIZE: 5.3578547 * X(3)**2 + .8356891 * X(1) * X(5) + 37.293239 * X(1) - 40792.141

SUBJECT TO: 0. $\leq$ 85.334407 +0056858 * X(2) * X(5) +.0006262 *X(1)* X(4) - .0022053 * X(3) * X(5) $\leq$ 92.

AND: 90. $\leq$ 80.51249 + .0071317 * X(2) * X(5) + .0029955 * X(1) * X(2) + .0021813 * X(3)**2 $\leq$ 110.

AND: 20. $\leq$ 9.300961 + .0047026 * X(3) * X(5) + .0012547 * X(1) * X(3) + .0019085 * X(3) * X(4) $\leq$ 25.

BOUNDS: 78. $\leq$ X(1) $\leq$ 102.

33. $\leq$ X(2) $\leq$ 45.

27. $\leq$ X(i) $\leq$ 45. , i = 3,...,5

INCREMENTS: (.001 , .001 , .001 , .001 , .001)

STARTING POINT: (78.62 , 33.44 , 31.07 , 44.15 , 35.32)

ALGORITHM	NUMBER OF FUNCTION/ CONSTRAINT EVALUATIONS	RESULT
GRG	236/236	-30501.315 = F(78.5445 , 33.44 , 30.6952 , 44.15 , 35.2458)
FLEX	684/21906	-30665.54 = F(78. , 33. , 29.9953 , 45. , 36.7758)
DMC	871/1035	-30661.929 = F(78. , 33. , 30.018 , 44.999 , 36.719)

PROBLEM: C-11

SOURCE: Himmelblau [32], problem 14. Colville [11], problem 5.

VARIABLES: 6

CONSTRAINTS: 4

MINIMIZE: see Appendix 2

SUBJECT TO: see Appendix 2

BOUNDS: 1. $\le$ X(i) $\le$ 30000. , i = 1,...,4

-500. $\le$ x(5) $\le$ 30000.

INCREMENTS: (.001 , .001 , .001 , .001 , .001 , .001)

STARTING POINT: (8000. , 3000. , 14000. , 2000. , 300. , 10.)

ALGORITHM	NUMBER OF FUNCTION/ CONSTRAINT EVALUATIONS	RESULT
GRG	109/109	258540.002 = F(9067.2894 , 3998.8284 , 14537.013 , 4192.7656 , 155.8287 , -84.3802)
FLEX	3316/72997	248694.6 = F(15639.23 , 4356.786 , 15999.52 , 3427.736 , 221.037 , -276.938)
DMC	3448/4756	247862.952 = F(13999.5 , 4000. , 15969.9 , 2756.56 , 209.542 , -231.954)

PROBLEM C-12

SOURCE: Himmelblau [32], problem 7. Colville [11], problem 8.

VARIABLES : 3

CONSTRAINTS: 14

MINIMIZE: see Appendix 2

SUBJECT TO: see Appendix 2

BOUNDS: $200. \leq X(1) \leq 2000.$

$1000. \leq X(2) \leq 16000.$

$1. \leq X(3) \leq 120.$

INCREMENTS: (.001 , .001 , .001)

STARTING POINTS: (1745. , 12000. , 110.)

ALGORITHM	NUMBER OF FUNCTION/ CONSTRAINT EVALUATIONS	RESULT
GRG	134/134	-870.659 = F(1744.5231 , 12000.2622 , 107.7669)
FLEX	372/4407	-1162.037 = F(1728.371 , 16000. , 98.1267)
DMC	386/431	-1162.036 = F(1728.370 , 16000. , 98.140)

PROBLEM: C-13

SOURCE: Unpublished, B. Famili, reinforced concrete bridge design.

VARIABLES: 5

CONSTRAINTS: 7

MINIMIZE: F = 2448. * X(1) * X(2) + 1224. * X(3) * X(4) + 7344. * X(5)

SUBJECT TO: .0435 - X(2) / X(1) $\leq$ 0.

AND: .00667 - X(4) / X(3) $\leq$ 0.

AND: 555.678*X(2) + 277.84*X(3) - 2.5*X(1)*X(2)**3 - .5*X(1)*X(2) * X(3)**2 - X(1) * X(2)**2 * X(3) - .0833 * X(3)**3 * X(4) $\leq$ 0.

AND: 7615.6 - .0833*X(3)**3*X(4) - 2.5 X(1)*X(2)**3 - .5*X(1) * X(2) * X(3)**2 - X(1) * X(2)**2 * X(3) $\leq$ 0.

AND: 395.92*X(2) + 197.96*X(3) - .5*X(1)*X(2)*X(3)**2 - 2.5 * X(1) * X(2)**3 - X(1) * X(2)**2 * X(3) - .0833 * X(3)**3 * X(4) $\leq$ 0.

AND: .0833*X(5)**2 + .0000283*X(1)**2 + .78*X(5) - .00948*X(1) $\leq$ 0.

AND: .0222 * X(2) + .0111 * X(3) - 1. $\leq$ 0.

*** VARIATION A ***

BOUNDS: 4. $\leq$ X(1) $\leq$ 20.

.3124 $\leq$ X(2) $\leq$ 2.

20. $\leq$ X(3) $\leq$ 80.

.3124 $\leq$ X(4) $\leq$ 1.

.1 $\leq$ X(5) $\leq$ 3.

INCREMENTS: (.001 , .001 , .001 , .001 , .001)

STARTING POINT: (18. , 1.8 , 60. , .7 , 2.)

ALGORITHM	NUMBER OF FUNCTION/ CONSTRAINT EVALUATIONS	RESULT
GRG	130/130	46156.583 = F(14.7648 , .6423 , 46.8366 , .3124 , .6853)
FLEX	408/13145	61586.68 = F(17.5910 , .7652 , 53.6286 , .3576 , .7028)
DMC	749/644	46684.291 = F(7.203 , 1.231 , 48.723 , .325 , .762)

*** VARIATION B ***

BOUNDS: as in variation A

INCREMENTS: (1. , .0625 , 1. , .0625 , .001)

STARTING POINT: as in variation A

ALGORITHM	NUMBER OF FUNCTION/ CONSTRAINT EVALUATIONS	RESULT
GRG/R/N1	136/136	NO FEASIBLE SOLUTION
G-2/P	299/299	86392.368 = F(15. , 2. , 20. , .3125 , .722)
DMC	424/496	47401.848 = F(7. , .9375 , 56. , .375 , .767)

PROBLEM C-14

SOURCE: Himmelbalu [32], problem 16.

VARIABLES: 9

CONSTRAINTS: 13

```
MINIMIZE: F = -.5*(X(1)*X(4) - X(2)*X(3) + X(3)*X(9) - X(5)*X(9) +
                X(5) * X(8) - X(6) * X(7)

SUBJECT TO: X(3)**2 + X(4)**2 - 1.  ≤ 0.

AND:          X(9)**2 - 1.  ≤ 0.

AND:          X(5)**2 + X(6)**2 - 1.  ≤ 0.

AND:          X(1)**2 + (X(2) - X(9))**2 - 1.  ≤ 0.

AND:          (X(1) - X(5))**2 + (X(2) - X(6))**2 - 1.  ≤ 0.

AND:          (X(1) - X(7))**2 + (X(2)  X(8))**2 - 1.  ≤ 0.

AND:          (X(3)  X(5))**2 + (X(4) - X(6))**2 - 1.  ≤ 0.

AND:          (X(3) - X(7))**2 + (X(4) - X(8))**2 - 1.  ≤ 0.

AND:          X(7)**2 + (X(8) - X(9))**2 - 1.  ≤ 0.

AND:          X(2) * X(3) - X(1) * X(4) ≤ 0.

AND:          -X(3) * X(9) ≤ 0.

AND:          X(5) * X(9) ≤ 0.

AND:          X(6) * X(7) - X(5) * X(8) ≤ 0.

BOUNDS: -1.  ≤ X(i) ≤ 2.   , i = 1,...,8

        0.  ≤ X(9) ≤ 2.

INCREMENTS: (.001 , .001 , .001 , .001 , .001 , .001 , .001 , .001 , .001)

STARTING POINT: (0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.)
```

ALGORITHM	NUMBER OF FUNCTION/ CONSTRAINT EVALUATIONS	RESULT
GRG	10/10	0.0 = F(0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.)
FLEX	1701/40049	-.866 = F(-.9890 , .1479 , -.6197 , -.7849 , -.9895 , .1442 , -.6226 , -.7826 , 0.)
DMC	1563/1957	-.865 = F(-.411 , -.211 , .573 , -.819 , -.423 , -.906 , .584 , -.112 , .899)

PROBLEM: C-15

SOURCE: Ragsdell and Phillips [55], optimal welded structure.

VARIABLES: 4

CONSTRAINTS: 5

MINIMIZE: see Appendix 2

SUBJECT TO: see Appendix 2

BOUNDS: $.125 \leq X(i) \leq 10.$, $i = 1,4$

$.125 \leq X(i) \leq 3.$, $i = 2,3$

INCREMENTS: (.001 , .001 , .001 , .001)

STARTING POINTS: (4. , 2. , 1. , 7.)

ALGORITHM	NUMBER OF FUNCTION/ CONSTRAINT EVALUATIONS	RESULT
GRG	345/345	2.381 = F(8.2915 , .2444 , .2444 , 6.2184)
FLEX	1125/12919	2.381 = F(8.2914 , .2444 , .2444 , 6.2181)
DMC	1388/1793	2.602 = F(8.281 , .245 , .189 , 9.017)

PROBLEM: C-16

SOURCE: Himmelblau [32], problem 22. U.S. Steel problem.

VARIABLES: 6

CONSTRAINTS: 4

MINIMIZE: 4.3*X(1) + 31.8*X(2) + 63.3*X(3) + 15.8*X(4) + 68.5*X(5) +
4.7 * X(6)

SUBJECT TO: 32.97 - 17.1*X(1) - 38.2*X(2) - 204.2*X(3) - 212.3*X(4) -
623.4*X(5) - 1495.5*X(6) + 169.*X(1)*X(3) + 3580.*X(3)*X(5) +
3810.*X(4)*X(5) + 18500.*X(4)*X(6) + 24300.*X(5)*X(6) ≤ 0.

AND: 25.12 - 17.9 * X(1) - 36.8 * X(2) - 113.9 * X(3) -
169.7 * X(4) - 337.8 * X(5) - 1385.2 * X(6) + 139. * X(1) *
X(3) + 2450. * X(4) * X(5) + 16600.* X(4) * X(6) +
17200. * X(5) * X(6) ≤ 0.

AND: -124.08 + 273.* X(2) + 70. * X(4) + 819.* X(5) -
26000.* X(4) * X(5) ≤ 0.

AND: -173.02 - 159.9 * X(1) + 311. * X(2) - 587.* X(4) -
391.* X(5) - 2198.* X(6) + 14000.* X(1) * X(6) ≤ 0.

BOUNDS: 0. ≤ X(1) ≤ .31
0. ≤ X(2) ≤ .046
0. ≤ X(3) ≤ .068
0. ≤ X(4) ≤ .042
0. ≤ X(5) ≤ .028
0. ≤ X(6) ≤ .0134

INCREMENTS: (.001 , .001 , .001 , .001 , .001 , .001)

STARTING POINT: (.212 , .043 , .065 , .033 , .018 , .012)

ALGORITHM	NUMBER OF FUNCTION/ CONSTRAINT EVALUATIONS	RESULT
GRG	133/133	4.071 = F(0. , 0. , .063 , 0. , 0. , .0134)
FLEX	48/5353	5.271 = F(.1771 , .0169 , .0576 , .0124 , .0010 , .0131)
DMC	328/598	5.494 = F(.054 , .028 , .066 , 0. , .002 , .012)

PROBLEM: C-17

SOURCE: Chanaratna [10], Reinforced Concrete Beam.

VARIABLES: 2

CONSTRAINTS: 1

MINIMIZE: see Appendix 2

SUBJECT TO: see Appendix 2

BOUNDS: $1. \le X(1) \le 77$

$12. \le X(2) \le 40.$

INCREMENTS: (1. , 1.)

STARTING POINT: (74. , 24.)

ALGORITHM	NUMBER OF FUNCTION/ CONSTRAINT EVALUATIONS	RESULT
GRG/R/N1	94/94	379.2 = F(65. , 16.)
G-2/P	159/159	381.96 = F(66. , 15.)
DMC	53/86	381.486 = F(67. , 14.)

PROBLEM: C-18

SOURCE: Chanaratna [10], Reinforced Concrete Beam.

VARIABLES: 2

CONSTRAINTS: 1

MINIMIZE: see Appendix 2
SUBJECT TO: see Appendix 2

BOUNDS: $1. \leq X(1) \leq 77.$

$12. \leq X(2) \leq 40.$

INCREMENTS: (1. , 1.)

STARTING POINT: (74. , 24.)

ALGORITHM	NUMBER OF FUNCTION/ CONSTRAINT EVALUATIONS	RESULT
GRG/R/N1	77/77	499.32 = F(63. , 17.)
G-2/P	118/118	499.32 = F(63. , 17.)
DMC	49/68	499.2 = F(65. , 16.)

PROBLEM: C-19

SOURCE: Gisvold and Moe [24], Hatch cover.

VARIABLES: 2

CONSTRAINTS: 4

MIMINIZE: see Appendix 2

SUBJECT TO: see Appendix 2

BOUNDS: $1. \leq X(1) \leq 20.$

$1. \leq X(2) \leq 4.$

INCREMENTS: (1. , 1.)

STARTING POINT: (7. , 3.)

ALGORITHM	NUMBER OF FUNCTION/ CONSTRAINT EVALUATIONS	RESULT
GRG/R/N1	68/68	109 = F(7. , 2.)
G-2/P	82/82	124.64 = F(5. , 4.)
DMC	21/30	109 = F(7. , 2.)

PROBLEM: C-20

SOURCE: Unpublished, R. L. Judd, Shell and Tube Condenser.

VARIABLES: 6

CONSTRAINTS: 5

MINIMIZE: see Appendix 2

SUBJECT TO: see Appendix 2

BOUNDS: $1. \leq X(i) \leq 6.$, $i = 1,2$

$2. \leq X(3)) \leq 5.$

$1. \leq X(i) \leq 100.$, $i = 4,5$

$100. \leq X(6) \leq 6200.$

INCREMENTS: (1. , 1. , 1. , 1. , 1. , 1.)

STARTING POINT: (5. , 3. , 4. , 4. , 5. , 800.)

ALGORITHM	NUMBER OF FUNCTION/ CONSTRAINT EVALUATIONS	RESULT
GRG/R/N1	1069/1069	1352.602 = F(6. , 6. , 2. , 3. , 37. , 799.)
G-2/P	145/145	1317.687 = F(6. , 4. , 3. , 2. , 25. , 684.)
DMC	405/751	1227.655 = F(6. , 5. , 2. , 3. , 30. , 612.)

PROBLEM: C-21

SOURCE: Chanaratna [10], wooden frame

VARIABLES: 2

CONSTRAINTS: 3

MINIMIZE: 1152. * X(1) + 864. * X(2)

SUBJECT TO: -(1.8 - 2.25 / X(1) - 5832. / ((12. + 5.33*X(2)**3 / X(1)**3)*
X(1)**2) $\leq$ 0.

AND: -(1.8 - 4.5/((8. + 3.56*X(2)**3/X(1)**3)*X(2)) - 5832/((12. +
5.33**X(2)**3 / X(1)**3) * X(2)**2 $\leq$ 0

AND: -1.8 - 4.5/((8. + 3.56*X(2)**3/X(1)**3)*X(2)) -(729. - 5832/
((12. +5.33 * X(2)**3 / X(1)**3) 8X(2)**2) $\leq$ 0

BOUNDS: 1. $\leq$ X(i) $\leq$ 100. , i= 1,2

INCREMENTS: (1. , 1.)

STARTING POINT; (30. , 30.)

ALGORITHM	NUMBER OF FUNCTION/ CONSTRAINT EVALUATIONS	RESULT
GRG/R/N1	78/78	NO FEASIBLE SOLUTION
G-2/P	183/183	20448 = F(2. , 21.)
DMC	74/85	24768 = F(2. , 26.)

PROBLEM: U-1

SOURCE; Glankwahmdee [25], problem 1. Adapted from Kuester and Mize [37].

VARIABLES: 2

MINIMIZE: F = -(3803.84 + 138. * X(1) + 239.92 * X(2) - 123.08 *

X(1)**2 - 203.64 * X(2)**2 - 182.25 * X(1) * X(2))

BOUNDS: $-100 \le X(i) \le 100$, i=1,2

INCREMENTS: (1. , 1.)

STARTING POINT: (30. , 10.)

*** VARIATION A ***

ALGORITHM	NUMBER OF FUNCTION EVALUATIONS	RESULT
COMPLEX	47	-3754.2 = F(-1. , 1.)
G-2	61	-3840.12 = F(0. , 1.)
DMC	96	-3840.12 = F(0. , 1.)

*** VARIATION B ***

BOUNDS: $-100 \le X(i) \le 100$, i = 1,2

INCREMENTS: (1. , 1.)

STARTING POINT: (10. , 30.)

ALGORITHM	NUMBER OF FUNCTION EVALUATIONS	RESULT
COMPLEX	57	-3587.7 = F(2. , 0.)
G-2	51	-3040.12 = F(0. , 1.)
DMC	89	-3040.12 = F(0. , 1.)

*** VARIATION C ***

BOUNDS: -100. $\leq$ X(i) $\leq$ 100. , i = 1,2

INCREMENTS: (.001 , .001)

STARTING POINT: (10. , 30.)

ALGORITHM	NUMBER OF FUNCTION EVALUATIONS	RESULT
NM	191	-3877.358 = F(.186115 , .555804)
G-2	186	-3877.358 = F(.186 , .506)
DMC	209	-3877.358 = F(.186 , .506)

*** VARIATION D ***

BOUNDS: -100. $\leq$ X(i) $\leq$ 100. , i = 1,2

INCREMENTS: (.001 , .001)

STARTING POINT: (30. 10.)

ALGORITHM	NUMBER OF FUNCTION EVALUATIONS	RESULT
NM	187	-3877.358 = F(.186133 , .505795)
G-2	233	-3877.358 = F(.186 , .506)
DMC	244	-3877.358 = F(.186 .506)

PROBLEM: U-2

SOURCE: Glankwahmdee [25], problem 2. Himmelblau [32], number 28.

VARIABLES: 2

MINIMIZE: F = (X(1)**2 + X(2) - 11.)**2 + (X(1) + X(2)**2) - 7.)**2

*** VARIATION A ***

BOUNDS: -100. $\leq$ X(i) $\leq$ 100. , i = 1,2

INCREMENTS: (1. , 1.)

STARTING POINT: (-10. , -10.)

ALGORITHM	NUMBER OF FUNCTION EVALUATIONS	RESULT
COMPLEX	43	26. = F(2. , 2.) = F(3. , 3.)
G-2	25	8. = F(-4. , -3.)
DMC	91	8. = F(-4 , -3.)

*** VARIATION B ***

BOUNDS: -100. $\leq$ X(i) $\leq$ 100. , i = 1,2

INCREMENTS: (1. , 1.)

STARTING POINT: (-10. , 10.)

ALGORITHM	NUMBER OF FUNCTION EVALUATIONS	RESULT
COMPLEX	37	50. = F(3. , -3.)
G-2	19	2. = F(-3. , 3.)
DMC	95	2. = F(-3. , 3.)

*** VARIATION C ***

BOUNDS: -100. $\leq$ X(i) $\leq$ 100. , i = 1,2

INCREMENTS: (1. , 1.)

STARTING POINT: (8. , 3.)

ALGORITHM	NUMBER OF FUNCTION EVALUATIONS	RESULT
COMPLEX	67	0. = F(3. , 2.)
G-2	39	0. = F(3. , 2.)
DMC	89	0. = F(3. , 2.)

*** VARIATION D ***

BOUNDS: -100. $\leq$ X(i) $\leq$ 100. , i = 1,2

INCREMENTS: (.001 , .001)

STARTING POINT: (-10. , -10.)

ALGORITHM	NUMBER OF FUNCTION EVALUATIONS	RESULT
NM	174	1.59E-8 = F(-3.7793 , -3.2732)
G-2	108	4.10E-5 = F(-3.780 , -3.284)
DMC	236	5.49E-6 = F(-3.779 , -3.283)
CMC	198	3.01E-6 = F(-3.7795 , -3.2834)

*** VARIATION E ***

BOUNDS: -100. $\leq$ X(i) $\leq$ 100. , i = 1,2

INCREMENTS: (.001 , .001)

STARTING POINT: (-10. , 10.)

ALGORITHM	NUMBER OF FUNCTION EVALUATIONS	RESULT
NM	172	7.86E-8 = F(3.00003 , 1.99993)
G-2	82	4.22E-6 = F(-2.805 , 3.131)
DMC	274	0. = F(3. , 2.)

*** VARIATION F ***

BOUNDS: $-100. \leq X(i) \leq 100.$, i = 1,2

INCREMENTS: (.001 , .001)

STARTING POINT: (8. , 3.)

ALGORITHM	NUMBER OF FUNCTION EVALUATIONS	RESULT
NM	190	1.47E-7 = F(2.99997 , 1.99994)
G-2	71	1.70E-5 = F(3. , 1.999)
DMC	335	0. = F(3. , 2.)

PROBLEM: U-3

SOURCE: Glankwahmdee [25], problem 3, problem U-2 scaled

VARIABLES: 2

MINIMIZE: F = ((. * X(1)**2 + 2. * X(2) - 11.)**2 + (3. * X(1) - 4.
X(2)**2 - 7.)**2

*** VARIATION A ***

BOUNDS: -100. $\le$ X(i) $\le$ 100. , i = 1,2

INCREMENTS: (1. , 1.)

STARTING POINT: (-10. -10.)

ALGORITHM	NUMBER OF FUNCTION EVALUATIONS	RESULT
COMPLEX	40	0. = F(1. , 1.)
G-2	38	0. = F(1. , 1.)
DMC	89	0. = F(1. , 1.)

*** VARIATION B ***

BOUNDS: -100. $\le$ X(i) $\le$ 100. , i = 1,2

INCREMENTS: (.001 , .001)

STARTING POINT: (-10. , -10)

ALGORITHM	NUMBER OF FUNCTION EVALUATIONS	RESULT
NM	197	2.10E-7 = F(.99997 , 1.00005)
G-2	282	6.80E-5 = F(1. , .999)
DMC	220	4.10E-5 = F(-1.260 , -1.642)

PROBLEM: U-1

SOURCE: Glankwahmdee [25], problem 4. Himmelblau [32], number 2.

Rosenbrock's function.

VARIABLES: 2

MINIMIZE: F = 100. * (X(2) - X(1)**2)**2 - (1. - X(1))**2

*** VARIATION A ***

BOUNDS: -100. ≤ X(i) ≤ 100. , i = 1,2

INCREMENTS: (1. , 1.)

STARTING POINT: (-12. , 10.)

ALGORITHM	NUMBER OF FUNCTION EVALUATIONS	RESULT
COMPLEX	134	9. = F(4. , 16.)
G-2	50	1. = F(2. , 4.)
DMC	93	9. = F(4. , 16.)

*** VARIATION B ***

BOUNDS: -10. ≤ X(i) ≤ 10. , i = 1,2

INCREMENTS: (.1 , .1)

STARTING POINT: (-1.2 , 1.)

ALGORITHM	NUMBER OF FUNCTION EVALUATIONS	RESULT
COMPLEX	70	.8 = F(.2 , 0.)
G-2	100	.8 = F(.2 , 0.)
DMC	145	0. = F(1. , 1.)

*** VARIATION C ***

BOUNDS: -100. $\leq$ X(i) $\leq$ 100. , i = 1,2

INCREMENTS: (1. , 1.)

STARTING POINT: (10. , 30.)

ALGORITHM	NUMBER OF FUNCTION EVALUATIONS	RESULT
COMPLEX	50	36. = F(-5. , 25.)
G-2	44	49. = F(-6. , 36.)
DMC	117	64. = F(9. , 81.)

*** VARIATION D ***

BOUNDS: -100 $\leq$ X(i) $\leq$ 100. , i = 1,2

INCREMENTS: (.001 , .001)

STARTING POINT: (-12. , 10)

ALGORITHM	NUMBER OF FUNCTION EVALUATIONS	RESULT
NM	546	8.05E-8 = F(1.0001 , 1.0002)
G-2	916	8.17E-5 = F(1.009 , 1.018)
DMC	1599	9.54 = F(3.5579 , 12.4858)
DMC	1137	0. = F(1. , 1.)

*** VARIATION E ***

BOUNDS: -10. $\leq$ X(i) $\leq$ 10. , i = 1,2

INCREMENTS: (.001 , .001)

STARTING POINT: (-1.2 1.)

ALGORITHM	NUMBER OF FUNCTION EVALUATIONS	RESULT
NM	348	4.70E-8 = F(.9998 , .9996)
G-2	314	3.17E-5 = F(.991 , .982)
CMC	415	2.24E-8 = F(1.0001 , 1.0003)
DMC	432	0. = F(1. , 1.)

*** VARIATION F ***

BOUNDS: $-100. \leq X(i) \leq 100.$, i = 1,2

INCREMENTS: (.001 , .001)

STARTING POINT: (10. , 30)

ALGORITHM	NUMBER OF FUNCTION EVALUATIONS	RESULT
NM	646	3.98E-8 = F(.99995 , .99988)
G-2	50	20.2 = F(5.497 , 30.225)
DMC	1085	0. = F(1. , 1.)

PROBLEM: U-5

SOURCE: Glankwahmdee [25], problem 5. Adapted from Himmelblau [32], number 31.

VARIABLES: 2

MINIMIZE: see Appendix 2

*** VARIATION A ***

BOUNDS: -100. $\leq$ X(i) $\leq$ 100. , i = 1,2

INCREMENTS: (1. , 1.)

STARTING POINT: (10. , 10.)

ALGORITHM	NUMBER OF FUNCTION EVALUATIONS	RESULT
COMPLEX	44	.84 = F(1. , 1.)
G-2	43	.84 = F(1. , 1.)
DMC	83	.84 = F(1. , 1.)

*** VARIATION B ***

BOUNDS: -100. $\leq$ X(i) $\leq$ 100. , i = 1,2

INCREMENTS: (.001 , .001)

STARTING POINT: (10. , 10.)

ALGORITHM	NUMBER OF FUNCTION EVALUATIONS	RESULT
NM	162	.169043 = F(1.7953 , 1.3779)
G-2	209	.169044 = F(1.796 , 1.378)
DMC	207	.169045 = F(1.795 , 1.378)

PROBLEM: U-6

SOURCE: Glankwahmdee [25], problem 6. Himmelblau [32], number 26.
Fletcher and Powell [20].

VARIABLES: 4

```
MINIMIZE: F = (X(1) + 10.  * X(2))**2 + 5.  * (X(3) - X(4))**2 + (X(2) - 2.  *
              X(3))**4 + 10.  * (X(1)  X(4))**4
```

*** VARIATION A ***

BOUNDS: -100. ≤ X(i) ≤ 100. , i = 1,...,4

INCREMENTS: (1. , 1. , 1. , 1.)

STARTING POINT: (9. , 5. , -6. , 8.)

ALGORITHM	NUMBER OF FUNCTION EVALUATIONS	RESULT
COMPLEX	215	690. = F(-12. , 1. , -1. , -12.)
G-2	140	0. = F(0. , 0. , 0. , 0.)
DMC	226	17. = F(1. , 0. , 1. , 1.)

*** VARIATION B ***

BOUNDS: -100. ≤ X(i) ≤ 100. , i = 1,...,4

INCREMENTS: (.001 , .001 , .001 , .001)

STARTING POINT: (9. , 5. , -6. , 8.)

ALGORITHM	NUMBER OF FUNCTION EVALUATIONS	RESULT
NM	704	1.41E-7 = F(.00476 , -.00044 , .0053 , .0054)
G-2	933	.01125 = F(.056 , -.005 , .034 , .034)
CMC	1930	11.28 = F(-.3521 , -.0525 , -.4775 , .4912)
DMC	2175	3.34E-7 = F(-.02 , .002 , -.01 , -.01)

*** VARIATION C ***

BOUNDS: $-100. \leq X(i) \leq 100.$, $i = 1,\ldots,4$

INCREMENTS: (.001 , .001 , .001 , .001)

STARTING POINT: (3. , -1. , 0. , 1.)

ALGORITHM	NUMBER OF FUNCTION EVALUATIONS	RESULT
NM	422	5.89E-8 = F(-.000848 , .000065 , -.000972 , -.001033)
DMC	1202	-1.46E-7 = F(.010 , -.001 , -.001 , -.001)

*** VARIATION D ***

BOUNDS: $-100. \leq X(i) \leq 100.$, $i = 1,\ldots,4$

INCREMENTS: (.001 , .001 , .001 , .001)

STARTING POINT: (10. , -10. , 10. , -10)

ALGORITHM	NUMBER OF FUNCTION EVALUATIONS	RESULT
NM	487	7.42E-7 = F(-.014031 , .001396 , -.013520 , -.013648)
DMC	2772	6.38E-7 = F(-.020 , .002 , -.003 , -.003)

PROBLEM: U-7

SOURCE: Glankwahmdee [25], problem 7. Adapted from Himmelblau [32], number 8. Wood's function.

VARIABLES: 4

MINIMIZE: F = 100. * (X(2) - X(1)**2)**2 + (1. - X(1))**2 +
90. * (X(4) - X(3)**2)**2 + (1. - X(3))**2 +
10.1 * ((X(2) - 1.)**2 + (X(4) - 1.)**2) +
19.8 * (X(2) - 1.) * (X(4) - 1.)

*** VARIATION A ***

BOUNDS: -100. $\leq$ X(i) $\leq$ 100. , i = 1,...,4

INCREMENTS: (1. , 1. , 1. , 1.)

STARTING POINT: (-9. , -3. , -9. , -3.)

ALGORITHM	NUMBER OF FUNCTION EVALUATIONS	RESULT
COMPLEX	220	0. = F(1. , 1. , 1. , 1.)
G-2	112	0. = F(1. , 1. , 1. , 1.)
DMC	215	8. = F(-1. , 1. , -1. , 1.)

*** VARIATION B ***

BOUNDS: $-100. \leq X(i) \leq 100.$, $i = 1,\ldots,4$

INCREMENTS: (.001 , .001 , .001 , .001)

STARTING POINT: (-9. , -3. , -9. , -3.)

ALGORITHM	NUMBER OF FUNCTION EVALUATIONS	RESULT
NM	1809	6.23E-8 = F(1.00006 , 1.00015 , .999931 , .999867)
G-2	2206	8.46E-5 = F(.996 , .992 , 1.003 , 1.006)
CMC	1271	7.84 = F(-.6283 , .3917 , -1.2215 , 1.4886)
DMC	4211	3.60E-6 = F(.999 , .998 , 1.001 , 1.002)

PROBLEM: U-8

SOURCE: Glankwahmdee [25], number 8.

VARIABLES: 5

MINIMIZE: see Appendix 2

*** VARIATION A ***

BOUNDS: -100. $\leq$ X(i) $\leq$ 100. , i = 1,...,5

INCREMENTS: (1. , 1. , 1. , 1. , 1.)

STARTING POINT: (0. , 0. , 0. , 0. , 1.)

ALGORITHM	NUMBER OF FUNCTION EVALUATIONS	RESULT
COMPLEX	200	-51. = F(1. , 1. , 1. , 1. , 1.)
G-2	360	-729 = F(0. , 11. , 23. , 17. , 7.)
DMC	94	-1. = F(0. , 0. , 0. , 1. , 1.)

*** VARIATION B ***

BOUNDS: -100. $\leq$ X(i) $\leq$ 100. , i = 1,...,5

INCREMENTS: (1. , 1. , 1. , 1. , 1.)

STARTING POINT: (15. , 17. , 20. , 25. , 61.)

ALGORITHM	NUMBER OF FUNCTION EVALUATIONS	RESULT
COMPLEX	368	-731. = F(0. , 13. , 24. , 18. , 6.)
G-2	463	-734. = F(0. , 12. , 21. , 16. , 5.)
DMC	724	-727 = F(0. , 11. , 24. , 17. , 8.)

```
                    ***  VARIATION C  ***

BOUNDS: -100.  < X(i) < 100.   , i = 1,...,5

INCREMENTS: (.001 , .001 , .001 , .001 , .001

STARTING POINT: (0.   , 0.   , 0.   , 0.   , 1.)

            NUMBER OF
            FUNCTION
ALGORITHM   EVALUATIONS   RESULT

  NM           677        -739.823 = F(-.2320 , 11.4891 , 22.2725 , 16.5397

                                     6.1145)

  G-2          1459       -739.823 = F(-.230 , 11.485 , 22.267 , 16.533 ,

                                     6.113)

  DMC          30579      -739.823 = F(-.232 , 11.489 , 22.273 , 16.540 ,

                                     6.115)


                    ***  VARIATION D  ***

BOUNDS: -100.  < X(i) < 100.   , i = 1,...,5

INCREMENTS: (.001 , .001 , .001 , .001 , .001)

STARTING POINT: (15.   , 17.   , 20.   , 25.   , 61.)

            NUMBER OF
            FUNCTION
ALGORITHM   EVALUATIONS   RESULT

  NM           763        -739.823 = F(-2320 , 11.4872 , 22.2730 , 16.5401 ,

                                     6.11469)

  G-2          2723       -739.823 = F(-.232 , 11.489 , 22.273 , 16.540 ,

                                     6.115)

  DMC          56172      -739.823 = F(-.233 , 11.489 , 22.273 , 16.541 ,

                                     6.115)
```

PROBLEM: U-9

SOURCE: Stoecker [71], example 8-2, Ammonia storage tank.

VARIABLES: 2

```
MINIMIZE: F = 400.  * X(1)**.9 - 1000.  + 22.*(EXP((-3950. /
          (X(2) + 460.))  + 11.86) - 14.7)**1.2 + 144.  *
          (80.  - X(2)) / X(1)
```

*** VARIATION A ***

BOUNDS: 1. $\leq$ X(i) $\leq$ 100. , i = 1,2

INCREMENTS: (.001 , .001)

STARTING POINT: (3. , 50.)

ALGORITHM	NUMBER OF FUNCTION EVALUATIONS	RESULT
NM	169	5339.2528 = F(5.9536 , 5.36496)
CMC	393	5392.6923 = F(6.2841 , 2.8239)
DMC	376	5339.2528 = F(5.954 , 5.362)

*** VARIATION B ***

BOUNDS: 1. $\leq$ X(i) $\leq$ 100. , i = 1,2

INCREMENTS: (1. , 1.)

STARTING POINT: (3. , 50.)

ALGORITHM	NUMBER OF FUNCTION EVALUATIONS	RESULT
G-2	61	5339.385 = F(6. , 5.)
DMC	87	5339.385 = F(6. , 5.)
SD/SECT	94	5339.385 = F(6. , 5.)

PROBLEM: U-10

SOURCE: S. Tuthill [74], Air Duct Design.

VARIABLES: 3

MINIMIZE: see Appendix 2

BOUNDS: 6. $\leq$ X(i) $\leq$ 24. , i = 1,...,3

INCREMENTS: (1. , 1. , 1.)

STARTING POINT: (18. , 18. , 18.)

ALGORITHM	NUMBER OF FUNCTION EVALUATIONS	RESULT
G-2	37	1435.827 = F(18. , 19. , 17.)
DMC	54	1435.098 = F(20. , 19. , 16.)
SD/SECT	64	1435.098 = F(20. , 19. , 16.)

PROBLEM: U-11

SOURCE: S. Tuthill [74], Air Duct Design.

VARIABLES: 3

MINIMIZE: see Appendix 2

BOUNDS: 6. $\leq X(i) \leq$ 35. , i = 1,...,3

INCREMENTS: (1. , 1. , 1.)

STARTING POINT: (25. , 25. , 25.)

ALGORITHM	NUMBER OF FUNCTION EVALUATIONS	RESULT
G-2	90	2283.594 = F(26. , 31. , 22.)
DMC	117	2271.999 = F(26. , 29. , 22.)
SD/SECT	66	2283.857 = F(25. , 28. , 20.)

PROBLEM: U-11

SOURCE: B. Turnill [74], Air Duct Design.

VARIABLES: 2

MINIMIZE: see Appendix 1

BOUNDS: 6. $\leq$ X(I) $\leq$ 35. , I = 1,2

INCREMENTS: (1. , 1.)

STARTING POINT: (20. , 20.)

ALGORITHM	NUMBER OF FUNCTION EVALUATIONS	RESULT
G-1	37	477.210 = F(16. , 14.
DMC	69	477.210 = F(16. , 14.)
BISECT	49	477.210 = F(16. , 14.)

AD-A092 435 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH F/G 12/1
A COMPOSITE ALGORITHM FOR MIXED INTEGER CONSTRAINED NONLINEAR O--ETC(U)
JAN 80 D B FOX
UNCLASSIFIED AFIT-CI-80-1D NL

3 OF 3
AD
A092435

END
DATE
FILMED
1-81
DTIC

PROBLEM: U-13

SOURCE: Eason and Fenton [15], Minimum inertia gear train.

VARIABLES: 2

MINIMIZE: F = .1 * (12. + X(1)**2 + (1. + X(2)**2 / X(1)**2 +

(X(1)**2 * X(2)**2 + 100.) / (X(1) * X(2))**4)

*** VARIATION A ***

BOUNDS: $.5 \leq X(i) \leq 3.$, i = 1,2

INCREMENTS: (.001 , .001)

STARTING POINT: (.5 , .5)

ALGORITHM	NUMBER OF FUNCTION EVALUATIONS	RESULT
NM	102	1.74415 = F(1.74352 , 2.02931)
CMC	176	1.74415 = F(1.7435 , 2.0297)
DMC	171	1.74415 = F(1.744 , 2.029)

*** VARIATION B ***

BOUNDS: $.5 \leq X(i) \leq 3.$, i = 1,2

INCREMENTS: (.1 , .1)

STARTING POINT: (.5 , .5)

ALGORITHM	NUMBER OF FUNCTION EVALUATIONS	RESULT
G-2	50	1.745 = F(1.7 , 2.)
DMC	52	1.746 = F(1.7 , 1.9)
SD/SECT	38	1.745 = F(1.7 , 2.)

PROBLEM: U-14

SOURCE: Rosenbrock and Storey [61], Heavy Water Plant.

VARIABLES: 3

MINIMIZE: see Appendix 2

BOUNDS: $1. \leq X(1) \leq 20.$

$250. \leq X(2) \leq 500.$

$223. \leq X(3) \leq 295.$

INCREMENTS: (1. , 1. , 1.)

STARTING POINT: (10. , 370. , 259.)

ALGORITHM	NUMBER OF FUNCTION EVALUATIONS	RESULT
G-2	123	19673.527 = F(7. , 319. , 258.)
DMC	129	19673.441 = F(7. , 320. , 258.)
SD/SECT	86	19673.441 = F(7. , 320. , 258.)

PROBLEM: U-15

SOURCE: Fletcher and Powell [20]. Helical valley.

VARIABLES: 3

MINIMIZE: see Appendix 2

BOUNDS: $-10 \le X(i) \le 10.$, $i = 1,2$

$-2.5 \le X(3) \le 7.5$

INCREMENTS: (.001 , .001 , .001)

STARTING POINT: (-1. , 0. , 0.)

ALGORITHM	NUMBER OF FUNCTION EVALUATIONS	RESULT
NM	365	1.07E-7 = F(1.00002 , -.00012 , -.00017)
CMC	936	1.269 = F(1.0542 , -.0081 , -.1103)
DMC	1398	0. = F(1. , 0. , 0.)

APPENDIX 2.

The purpose of this appendix is to insure that results from this research can be reproduced. The FORTRAN function subprograms used to compute the objective function (and for constrained problems the constraint functions) are included here. If there should be inadvertent disagreement between the problem description in Appendix 1 and here in Appendix 2 then Appendix 2 should be considered authoratative.

In every function subprogram the objective function value is computed following FORTRAN statement number 1000. To obtain this value the function subprogram is called with the parameter K in the calling sequence equal to 0. To evaluate constraints, in constrained problems, the function is called with K set to 1, 2, ..., m where m is the number of constraints. The parameter K is used in the COMPUTED GO TO statement in order to branch to the approprate code to calculate the specified constraint function. Thus the evaluation of m constraint functions and the objective function requires m+1 calls to the FORTRAN function subprogram.

Problems labeled C-1 to C-21 are the constrained test problems. Problems labeled U-1 to U-18 are the unconstrained test problems.

PROBLEM	FUNCTION NAME	PAGE	PROBLEM	FUNCTION NAME	PAGE
C-1	ALKY	190	U-1	AG1	210
C-2	BART	191	U-2	AG2	210
C-3	BEAR1	192	U-3	AG3	210
C-4	BOXA	192	U-4	AG4	211
C-5	BOXB	193	U-5	AG5	211
C-6	FLY	194	U-6	AG6	212
C-7	POST	195	U-7	AG7	212
C-8	CHEM1	195	U-8	AG8	212
C-9	COL1	196	U-9	AMMO	213
C-10	COL3	197	U-10	DUCT1	214
C-11	COL5	198	U-11	DUCT4	218
C-12	COL8	199	U-12	DUCT9	219
C-13	FAM2	200	U-13	GEAR1	221
C-14	RAC	201	U-14	WATER	222
C-15	RP	202	U-15	POW2	222
C-16	STEEL	204	U-16	OBJT3	223
C-17	CH3	204	U-17	OBJT4	224
C-18	CH3B	205	U-18	OBJT5	224
C-19	GM1	206			
C-20	STEAM	206			
C-21	CH1	209			

Problem: C-1

```
      REAL FUNCTION ALKY(X,K)
      COMMON/PRINT/IBRKT,IPOW,IMS,JMSP,IIN,IOUT,IOTT
      DIMENSION X(20)
      LOGICAL FLAG
      DATA FLAG/.FALSE./
      IF(FLAG) GO TO 500
      FLAG = .TRUE.
C      ONE TIME CALCULATIONS HERE
      C1 = .063
      C2 = 5.04
      C3 = .035
      C4 = 10.
      C5 = 3.36
      D4L = .99
      D4U = 1./D4L
      D7L = D4L
      D7U = D4U
      D9L = .9
      D9U = 1./D9L
      D10L = D4L
      D10U = D4U
      PRINT(IOUT,*)" ALKYLATION PROCESS"
500    CONTINUE
C      EVERYTIME CALCULATIONS HERE
      X1 = X(1)
      X2 = X(2)
      X3 = X(3)
      X4 = X(4)
      X7 = X(5)
      X9 = X(6)
      X10 = X(7)
      X5 = 1.22 * X4 - X1
      X6 = (98000.*X3) / (X3* 1000. + X4 * X9)
      X8 = (X2 + X5) / X1
      IF(K .EQ. 0) GO TO 1000
      GO TO (1,2,3,4,5,6,7),K
1     CONTINUE
C     _ CONSTRAINTS HERE
      C = X1* (1.12 + .13167*X8 - .00667*X8**2)
      ALKY = D4L*X4 - C
      IF(ALKY .LT. 0.) ALKY  = C - D4U*X4
      RETURN
2     CONTINUE
      C = 86.35 + 1.089 * X8 - .038 * X8**2 + .325*(X6 - 89.)
      ALKY = D7L * X7 - C
      IF(ALKY .LT. 0.) ALKY = C - D7U * X7
      RETURN
3     CONTINUE
      C = 35.82 - .222 * X10
      ALKY = D9L * X9 - C
      IF(ALKY .LT. 0.) ALKY = C - D9U * X9
      RETURN
4     CONTINUE
      C = 3. * X7 - 133.
      ALKY = D10L * X10 - C
      IF(ALKY .LT. 0.) ALKY = C - D10U * X10
      RETURN
5     CONTINUE
      ALKY = X5 - 2000.
      IF(ALKY .LT. 0.)ALKY = 1. - X5
      RETURN
6     CONTINUE
       ALKY = X8 - 12.
      IF(ALKY .LT. 0.) ALKY = 3. - X8
      RETURN
7     CONTINUE
      ALKY = 85. - X6
      IF(ALKY .LT. 0.) ALKY = X6 - 93.
      RETURN
1000  CONTINUE
      ALKY = -(C1 * X4 * X7 - C2 * X1 - C3 * X2 - C4 * X3 - C5 * X5)
      RETURN
      END
```

Problem: C-2

```
      REAL FUNCTION BART(X,K)
      COMMON/PRINT/IBRKT,IPOW,IMS,JMSP,IIN,IOUT,IOTT
      DIMENSION X(20)
      LOGICAL FLAG
      DATA FLAG/.FALSE./
      IF(FLAG) GO TO 500
      PRINT(IOUT,*)" R.L. FOX 2 BAR TRUSS"
      FLAG = .TRUE.
C      ONE TIME CALCULATIONS HERE
      P = 33.
      B = 30.
      T = .1
      E = 30000.
      PI = 3.14159
500    CONTINUE
C      EVERYTIME CALCULATIONS HERE
      H = X(1)
      D = X(2) / 4.
      IF(K .EQ. 0) GO TO 1000
      GO TO (1,2),K
1     CONTINUE
C     _ CONSTRAINTS HERE
      F = (P/(PI*T))*SQRT(B**2 + H**2)/(H*D)
      BART = F - 100.
      RETURN
2     CONTINUE
      BART = F - PI**2 * E * (D**2 + T**2)/(8.*(B**2+H**2))
      RETURN
1000  CONTINUE
      BART = .6 * PI * D * T * SQRT(B**2 + H**2)
      RETURN
      END
```

Problem: C-3

```
      REAL FUNCTION BEAR1(X,K)
      COMMON/PRINT/IBRKT,IPOW,IMS,JMSP,IIN,IOUT,IOTT
      DIMENSION X(20)
      LOGICAL FLAG
      DATA FLAG/.FALSE./
      IF(FLAG) GO TO 500
      PRINT(IOUT,*)" EASON AND FENTON 6: JOURNAL BEARING DESIGN"
      FLAG = .TRUE.
500    CONTINUE
      IF(K .EQ. 0) GO TO 1000
1     CONTINUE
C     _ CONSTRAINTS HERE
      BEAR1 = 8.62 * X(2)**3 / X(1) - 1.
      RETURN
1000  CONTINUE
      BEAR1 = (.44*X(1)**3 / X(2)**2 + 10./X(1) + .592*X(1)/X(2)**3)/10.
      RETURN
      END
```

Problem: C-4

```
      REAL FUNCTION BOXA(X,K)
      COMMON/PRINT/IBRKT,IPOW,IMS,JMSP,IIN,IOUT,IOTT
      DIMENSION X(20)
      LOGICAL FLAG
      DATA FLAG/.FALSE./
      IF(FLAG) GO TO 500
      PRINT(IOUT,*)" BOX PROBLEM A,  HIMMELBLAU NO. 13"
      FLAG = .TRUE.
C      ONE TIME CALCULATIONS HERE
      C0 = -24345.
      C1 = -8720288.849
      C2 = 150512.5253
      C3 = -156.6950325
      C4 = 476470.3222
      C5 = 729482.8271
      C6 = -145421.402
      C7 = 2931.1506
      C8 = -40.427932
      C9 = 5106.192
      C10 = 15711.36
      C11 = -155011.1084
      C12 = 4360.53352
      C13 = 12.9492344
      C14 = 10236.884
      C15 = 13176.786
      C16 = -326669.5104
      C17 = 7390.68412
      C18 = -27.8986976
      C19 = 16643.076
      C20 = 30988.146
500    CONTINUE
      IF(K .EQ. 0) GO TO 1000
      GO TO (1,2,3),K
1     CONTINUE
C     _ CONSTRAINTS HERE
      X6 = C6 * X(1) + C7 * X(1) * X(2) + C8 * X(1) * X(3) +
     1  C9 * X(1) * X(4) + C10 * X(1) * X(5)
      BOXA = -X6
      IF(BOXA .LT. 0.) BOXA = X6 - 294000.
      RETURN
2     CONTINUE
      X7 = C11 * X(1) + C12 * X(1) * X(2) + C13 * X(1) * X(3) +
     1  C14 * X(1) * X(4) + C15 * X(1) * X(5)
      BOXA = -X7
      IF(BOXA .LT. 0.) BOXA = X7 - 294000.
      RETURN
3     CONTINUE
      X8 = C16 * X(1) + C17 * X(1) * X(2) + C18 * X(1) * X(3) +
     1  C19 * X(1) * X(4) + C20 * X(1) * X(5)
      BOXA = -X8
      IF(BOXA .LT. 0.) BOXA = X8 - 277200.
      RETURN
1000  CONTINUE
      BOXA =(C0 + C1 * X(1) + C2 * X(1) * X(2) + C3 * X(1) * X(3) +
     1  C4 * X(1) * X(4) + C5 * X(1) * X(5))*(-.000001)
      RETURN
      END
```

Problem: C-5

```
      REAL FUNCTION BOXB(X,K)
      COMMON/PRINT/IBRKT,IPOW,IMS,JMSP,IIN,IOUT,IOTT
      DIMENSION X(20)
      LOGICAL FLAG
      DATA FLAG/.FALSE./
```

```
        DATA SQR3/1.732050808/
       IF(FLAG) GO TO 500
       PRINT(IOUT,*)"BOX PROBLEM B"
       FLAG = .TRUE.
500     CONTINUE
       IF(K .EQ. 0) GO TO 1000
       GO TO (1,2),K
1      CONTINUE
       B = X(1) + SQR3 * X(2)
       BOXB = -B
       IF(BOXB .LT. 0.) BOXB = B - 6.
       RETURN
2      CONTINUE
       BOXB = X(2) - X(1) /SQR3
       RETURN
C      _ CONSTRAINTS HERE
1000   CONTINUE
       BOXB = -((9. - (X(1) - 3.) **2) * X(2)**3/(27. * SQR3) )
       RETURN
       END
```

Problem: C-6

```
       REAL FUNCTION FLY(X,K)
       COMMON/PRINT/IBRKT,IPOW,IMS,JMSP,IIN,IOUT,IOTT
       DIMENSION X(20)
       LOGICAL FLAG
       DATA FLAG/.FALSE./
       IF(FLAG) GO TO 500
       PRINT(IOUT,*)" EASON & FENTON 7: FLYWHEEL DESIGN"
       FLAG = .TRUE.
500     CONTINUE
       IF(K .EQ. 0) GO TO 1000
       GO TO (1,2),K
1      CONTINUE
C      _ CONSTRAINTS HERE
       FLY = X(1)**2 * X(2) - 675.
       RETURN
2      CONTINUE
       FLY = (X(1) * X(3))**2/1.E7 - .419
       RETURN
1000   CONTINUE
       FLY = (-.0201 * X(1)**4 * X(2) * X(3)**2)/1.E7
       RETURN
       END
```

Problem: C-7

```
      REAL FUNCTION POST(X,K)
      COMMON/PRINT/IBRKT,IPOW,IMS,JMSP,IIN,IOUT,IOTT
      DIMENSION X(20)
      LOGICAL FLAG
      DATA FLAG/.FALSE./
      IF(FLAG) GO TO 500
      PRINT(IOUT,*)" EASON & FENTON 2: MAX SIZE POST BOX"
      FLAG = .TRUE.
500    CONTINUE
      IF(K .EQ. 0) GO TO 1000
1     CONTINUE
C     _ CONSTRAINTS HERE
      C = X(1) + 2. * (X(2) + X(3))
      POST = -C
      IF(POST .LT. 0.) POST = C - 72.
      RETURN
1000  CONTINUE
      POST = - X(1) * X(2) * X(3) * .001
      RETURN
      END
```

Problem: C-8

```
      REAL FUNCTION CHEM1(X,K)
      COMMON/PRINT/IBRKT,IPOW,IMS,JMSP,IIN,IOUT,IOTT
      DIMENSION X(20)
      DIMENSION Y(10),C(10)
      LOGICAL FLAG
      DATA FLAG/.FALSE./
      DATA C/-6.089 , -17.164 , -34.054 , -5.914 , -24.721
     1 , -14.986 , -24.1 , -10.708 , -26.662 , -22.179/
      IF(FLAG) GO TO 500
      PRINT(IOUT,*)" CHEMICAL EQUILIBRIUM, HIMMELBLAU NO. 4"
      FLAG = .TRUE.
500    CONTINUE
C      EVERYTIME CALCULATIONS HERE
      DO 901 I = 1,6
901   Y(I) = X(I)
      Y(8) = X(7)
      Y(7) = 1. - Y(4) - 2.*Y(5) - Y(6)
      Y(10) = 2. - Y(1) - 2. * Y(2) - 2. * Y(3) - Y(6)
      Y(9) = (1. - Y(10) - Y(3) - Y(7) - Y(8))/2.
      IF(K .EQ. 0) GO TO 1000
      GO TO (1,2,3),K
1     CONTINUE
C     _ CONSTRAINTS HERE
      CHEM1 = -Y(7)
      IF(CHEM1 .LT. 0.) CHEM1 = Y(7) - 1.
      RETURN
2     CONTINUE
```

```
      CHEM1 = - Y(10)
      IF(CHEM1 .LT. 0.) CHEM1 = Y(10) - 1.
      RETURN
3     CONTINUE
      CHEM1 = -Y(9)
      IF(CHEM1 .LT. 0.) CHEM1 = Y(9) - .5
      RETURN
1000  CONTINUE
      SUM = 0.
      DO 20 I = 1,10
20    SUM = SUM + Y(I)
      F = 0.
      DO 30 I = 1,10
      IF(Y(I) .LT. 1.E-9) GO TO 30
      F = F + Y(I) * (C(I) + ALOG(Y(I) / SUM))
30    CONTINUE
      CHEM1 = F
      RETURN
      END
```

Problem: C-9

```
      REAL FUNCTION COL1(X,K)
      COMMON/PRINT/IBRKT,IPOW,IMS,JMSP,IIN,IOUT,IOTT
      DIMENSION X(20)
      DIMENSION E(5) , C(5,5) , D(5) , A(10,5) , B(10)
      LOGICAL FLAG
      DATA FLAG/.FALSE./
      IF(FLAG) GO TO 500
      DATA E/-15. , -27. , -36. , -18. , -12./
      DATA (C(1,J),J=1,5)/30. , -20. , -10. , 32. , -10./
      DATA (C(2,J),J=1,5)/-20., 39. , -6. , -31., 32./
      DATA (C(3,J),J=1,5)/-10., -6. , 10. , -6. , -10./
      DATA (C(4,J),J=1,5)/32. , -31. , -6. , 39. , -20./
      DATA (C(5,J),J=1,5)/-10., 32. , -10. ,-20. , 30./
      DATA D/4. , 8. , 10. , 6. , 2./
      DATA (A(1,J),J=1,5)/-16. ,   2. ,   0. ,   1. ,   0./
      DATA (A(2,J),J=1,5)/  0. ,  -2. ,   0. ,   .4 ,   2./
      DATA (A(3,J),J=1,5)/-3.5 ,   0. ,   2. ,   0. ,   0./
      DATA (A(4,J),J=1,5)/  0. ,  -2. ,   0. ,  -4. ,  -1./
      DATA (A(5,J),J=1,5)/  0. ,  -9. ,  -2. ,   1. , -2.8/
      DATA (A(6,J),J=1,5)/  2. ,   0. ,  -4. ,   0. ,   0./
      DATA (A(7,J),J=1,5)/-1. ,  -1. ,  -1. ,  -1. ,  -1./
      DATA (A(8,J),J=1,5)/-1. ,  -2. ,  -3. ,  -2. ,  -1./
      DATA (A(9,J),J=1,5)/1. ,    2. ,   3. ,   4. ,   5./
      DATA (A(10,J),J=1,5)/1. ,   1. ,   1. ,   1. ,   1./
      DATA B/-40. , -2. , -.25 , -4. , -4. , -1. ,
     1 -40. , -60. , 5. , 1./
      PRINT(IOUT,*)" COLVILLE 1"
      FLAG = .TRUE.
```

```
500     CONTINUE
       IF(K .EQ. 0) GO TO 1000
1      CONTINUE
C      _ CONSTRAINTS HERE
       C1 = 0.
       DO 110 J = 1,5
       C1 = C1 + A(K,J)*X(J)
110    CONTINUE
       COL1 = B(K) - C1
       RETURN
1000   CONTINUE
       F1 = 0.
       F2 = 0.
       F3 = 0.
       DO 250 J = 1,5
       F1 = F1 + E(J) * X(J)
       F3 = F3 + D(J) * X(J)**3
       DO 220 I = 1,5
       F2 = F2 + C(I,J) * X(J) * X(I)
220    CONTINUE
250     CONTINUE
       COL1 = F1 + F2 + F3
       RETURN
       END
```

Problem: C-10

```
       REAL FUNCTION COL3(X,K)
       COMMON/PRINT/IBRKT,IPOW,IMS,JMSP,IIN,IOUT,IOTT
       DIMENSION X(20)
       LOGICAL FLAG
       DATA FLAG/.FALSE./
       IF(FLAG) GO TO 500
       PRINT(IOUT,*)" COLVILLE  3"
       FLAG = .TRUE.
500     CONTINUE
       IF(K .EQ. 0) GO TO 1000
       GO TO (1,2,3),K
1      CONTINUE
C      _ CONSTRAINTS HERE
       C = 85.334407 + .0056858* X(2)*X(5) + .0006262* X(1)*X(4)
      1 - .0022053 * X(3)*X(5)
       COL3 = -C
       IF(COL3 .LT. 0.) COL3 = C-92.
       RETURN
2      CONTINUE
       C = 80.51249 + .0071317 * X(2) * X(5) + .0029955 * X(1) * X(2)
      1 + .0021813 * X(3)**2
       COL3 = -(C - 90.)
       IF(COL3 .LT. 0.) COL3 = C-110.
      RETURN
3     CONTINUE
      C = 9.300961 + .0047026 * X(3) * X(5) + .0012547 * X(1) * X(3)
     1  + .0019085 * X(3) * X(4)
      COL3 = -(C-20.)
      IF(COL3 .LT. 0.) COL3 = C-25.
      RETURN
1000  CONTINUE
      COL3 = 5.3578547 * X(3)**2 + .8356891 * X(1)*X(5)
     1  + 37.293239 * X(1) - 40792.141
      RETURN
      END
```

Problem: C-11

```
      REAL FUNCTION COL5(X,K)
      COMMON/PRINT/IBRKT,IPOW,IMS,JMSP,IIN,IOUT,IOTT
      DIMENSION X(20)
      DIMENSION W(6)
      LOGICAL FLAG
      DATA FLAG/.FALSE./
      DATA W/4*1. , 2* 100./
      IF(FLAG) GO TO 500
      PRINT(IOUT,*)" COLVILLE 5"
      FLAG = .TRUE.
500    CONTINUE
      IF(K .EQ. 0) GO TO 1000
      GO TO (1,2,3,4),K
1     CONTINUE
C     _ CONSTRAINTS HERE
      CAPT1 = (.0285* X(1) + 300.) / (.1425E-3 * X(1) + 1.)
      T1 = 500. - CAPT1
      T3 = FCALL(T1 , 350. , .915 , .936E-4 , X(3) )
      COL5 = -(T3 - 300.)
      RETURN
2     CONTINUE
      T2 = FCALL(200. , 300. , 1.5 , .333E-3 , X(2) )
      T4 = FCALL(T2 , 350. , .8 , 1.25E-3 , X(4) )
      COL5 = -(T4 - 300.)
      RETURN
3     CONTINUE
      CAPT1 = (.0285* X(1) + 300.) / (.1425E-3 * X(1) + 1.)
      CAPT2 = FCAPH(200. , 300. , 1.5 , .333E-3 , X(2) )
      CAPTJ1 = .7*CAPT1 + .3 * CAPT2
      CAPT6 = FCAPH(80. , CAPTJ1 , 0. , 3.E-4 , X(6) )
      COL5 = CAPT6 - 250.
      RETURN
4     CONTINUE
      CAPT1 = (.0285* X(1) + 300.) / (.1425E-3 * X(1) + 1.)
      T1 = 500. - CAPT1
      T2 = FCALL(200. , 300. , 1.5 , .333E-3 , X(2) )
      CAPT4 = FCAPH(T2 , 350. , .8 , 1.25E-3 , X(4) )
      CAPT3 = FCAPH(T1 , 350. , .915 , .936E-4 , X(3) )
      CAPTJ2 = .8 * CAPT3 + .2 * CAPT4
      CAPT5 = FCAPH(80. , CAPTJ2 , 0. , 3.75E-4 , X(5) )
      COL5 = CAPT5 - 280.
      RETURN
1000  CONTINUE
      F = 0.
      DO 40 I = 1,6
      Z = 0.
      IF(X(I) .LT. 0.) GO TO 35
      Y = X(I)/2000.
      Z = FLOAT(IFIX(Y))
      IF(Z .EQ. Y) GO TO 35
      Z = Z+1
35    CONTINUE
      F = F + (2.7 * X(I) + 1300. * Z) * W(I)
40    CONTINUE
      COL5 = F
      RETURN
      END
```

Problem: C-12

```
      REAL FUNCTION COL8(X,K)
      COMMON/PRINT/IBRKT,IPOW,IMS,JMSP,IIN,IOUT,IOTT
      DIMENSION X(20)
      LOGICAL FLAG
      DATA FLAG/.FALSE./
      IF(FLAG) GO TO 500
      PRINT(IOUT,*)" COLVILLE 8  PROCESS OPT."
      FLAG = .TRUE.
500    CONTINUE
      IF(K .EQ. 0) GO TO 1000
      GO TO (1,2,3,4,5,6,7),K
1     CONTINUE
C     _ CONSTRAINTS HERE
      Y2 = 1.6 * X(1)
110   Y3 = 1.22 * Y2 - X(1)
      Y6 = (X(2) + Y3) / X(1)
      Y2CALC = X(1) * (112. + 13.167 * Y6 - .6667 * Y6**2) * .01
      IF(ABS(Y2CALC - Y2) - .001) 130,130,120
120   Y2 = Y2CALC
      GO TO 110
130   CONTINUE
      COL8 = - Y2
      IF(COL8 .LT. 0.) COL8 = Y2 - 5000.
      RETURN
2     CONTINUE
```

```
      COL8 = -Y3
      IF(COL8 .LT. 0.) COL8 = Y3 - 2000.
      RETURN
3     CONTINUE
      Y4 = 93.
1100  Y5 = 86.35 + 1.098 * Y6 - .038 * Y6**2 + .325 * (Y4 - 89.)
      Y8 = -133. + 3. * Y5
      Y7 = 35.82 - .222 * Y8
      Y4CALC = 98000. * X(3) / (Y2 * Y7 + X(3) * 1000.)
      IF(ABS(Y4CALC - Y4) - .0001) 1300,1300,1200
1200  Y4 = Y4CALC
      GO TO 1100
1300  CONTINUE
      COL8 = 85. - Y4
      IF(COL8 .LT. 0.) COL8  = Y4 - 93.
      RETURN
4     CONTINUE
      COL8 = 90. - Y5
      IF(COL8 .LT. 0.) COL8 = Y5 - 95.
      RETURN
5     CONTINUE
      COL8 = 3. - Y6
      IF(COL8 .LT. 0.) COL8 = Y6 - 12.
      RETURN
6     CONTINUE
      COL8 = .01 - Y7
      IF(COL8 .LT. 0.) COL8 = Y7 - 4.
      RETURN
7     CONTINUE
      COL8 = 145. - Y8
      IF(COL8 .LT. 0.) COL8 = Y8 - 162.
      RETURN
1000  CONTINUE
      COL8 = -(.063 * Y2 * Y5 - 5.04 * X(1) - 3.36 * Y3 -
     1   .035 * X(2) - 10. * X(3) )
      RETURN
      END
```

Problem: C-13

```
      REAL FUNCTION FAM2(X,K)
      COMMON/PRINT/IBRKT,IPOW,IMS,JMSP,IIN,IOUT,IOTT
      DIMENSION X(20)
      LOGICAL FLAG
      DATA FLAG/.FALSE./
      IF(FLAG) GO TO 500
      PRINT(IOUT,*)" FAMILI BRIDGE DESIGN"
      FLAG = .TRUE.
500    CONTINUE
      IF(K .EQ. 0) GO TO 1000
      GO TO (1,2,3,4,5,6,7),K
1     CONTINUE
      FAM2 = .0435 - X(2) / X(1)
      RETURN
2     CONTINUE
      FAM2 = .00667 - X(4) / X(3)
      RETURN
C     _ CONSTRAINTS HERE
3     CONTINUE
      FAM2 = 555.678  * X(2) + 277.84 * X(3) - 2.5 *
     1 X(1) * X(2)**3 - .5 * X(1) * X(2) * X(3)**2 -
     2 X(1) * X(2)**2 * X(3) - .0833 * X(3)**3 * X(4)
      RETURN
4     CONTINUE
      FAM2 = 7615.6 - .0833 * X(3)**3 * X(4) - 2.5 * X(1) *
     1 X(2)**3 - .5 * X(1) * X(2) * X(3)**2 -
     2 X(1) * X(2)**2 * X(3)
      RETURN
5     CONTINUE
      FAM2 = 395.92  * X(2) + 197.96 * X(3) - .5 * X(1) *
     1 X(2) * X(3)**2 - 2.5 * X(1) * X(2)**3 - X(1) *
     2 X(2)**2 * X(3) - .0833 * X(3)**3 * X(4)
      RETURN
6     CONTINUE
      FAM2 = .0833 * X(5)**2 + .0000283 * X(1)**2 + .78 - X(5) -
     1 .00948 * X(1)
      RETURN
7     CONTINUE
      FAM2 = .0222  * X(2) + .0111 * X(3) - 1.
      RETURN
1000  CONTINUE
      FAM2 = 2448. * X(1) * X(2) + 1224. * X(3) * X(4) +
     1 7344. * X(5)
      RETURN
      END
```

Problem: C-14

```
      REAL FUNCTION RAC(X,K)
      COMMON/PRINT/IBRKT,IPOW,IMS,JMSP,IIN,IOUT,IOTT
      DIMENSION X(20)
      LOGICAL FLAG
      DATA FLAG/.FALSE./
      IF(FLAG) GO TO 500
      PRINT(IOUT,*)" RAC TP 302,  HIMMELBALU NO. 16"
      FLAG = .TRUE.
500    CONTINUE
      IF(K .EQ. 0) GO TO 1000
      GO TO (1,2,3,4,5,6,7,8,9,10,11,12,13),K
1     CONTINUE
```

```
C       _ CONSTRAINTS HERE
        RAC = X(3)**2 + X(4)**2 - 1.
        RETURN
2       CONTINUE
        RAC = X(9)**2 - 1.
        RETURN
3       CONTINUE
        RAC = X(5)**2 + X(6)**2 - 1.
        RETURN
4       CONTINUE
        RAC = X(1)**2 + (X(2) - X(9))**2 - 1.
        RETURN
5       CONTINUE
        RAC = (X(1) - X(5))**2 + (X(2) - X(6))**2 - 1.
        RETURN
6       CONTINUE
        RAC = (X(1) - X(7))**2 + (X(2) - X(8))**2 - 1.
        RETURN
7       CONTINUE
        RAC = (X(3) - X(5))**2 + (X(4) - X(6))**2 - 1.
        RETURN
8       CONTINUE
        RAC = (X(3) - X(7))**2 + (X(4) - X(8))**2 - 1.
        RETURN
9       CONTINUE
        RAC = X(7)**2 + (X(8) - X(9))**2 - 1.
        RETURN
10       CONTINUE
        RAC = X(2) * X(3) - X(1) * X(4)
        RETURN
11      CONTINUE
        RAC = -X(3) * X(9)
        RETURN
12      CONTINUE
        RAC = X(5) * X(9)
        RETURN
13      CONTINUE
        RAC = X(6) * X(7) - X(5) * X(8)
        RETURN
1000    CONTINUE
        RAC = -.5 * (X(1)*X(4) - X(2)*X(3) + X(3)*X(9) - X(5)*X(9)
       1  + X(5)*X(8) - X(6)*X(7))
        RETURN
        END
```

Problem: C-15

```
        REAL FUNCTION RP(X,K)
        COMMON/PRINT/IBRKT,IPOW,IMS,JMSP,IIN,IOUT,IOTT
        DIMENSION X(20)
```

```
      LOGICAL FLAG
      DATA FLAG/.FALSE./
      IF(FLAG) GO TO 500
      PRINT(IOUT,*)" RAGSDELL AND PHILLIPS, OPT WELD STRUCT"
      CAPF = 6000.
      CAPL = 14.
      CAPE = 30.E6
      CAPG = 12.E6
      FLAG = .TRUE.
C      ONE TIME CALCULATIONS HERE
500    CONTINUE
      X3 = X(1)
      X4 = X(2)
      X1 = X(3)
      X2 = X(4)
      IF(K .EQ. 0) GO TO 1000
      GO TO (1,2,3,4,5),K
1     CONTINUE
C     _ CONSTRAINTS HERE
      SIGMA = (6. * CAPF * CAPL) / (X4 * X3 * X3)
      RP = SIGMA - 30000.
      RETURN
2     CONTINUE
      CAPI = (X3 * X4 **3) / 12.
      ALPHA = (CAPG * X3 * X4**3 ) / 3.
      PC = (4.013 * SQRT(CAPE*CAPI*ALPHA))/(CAPL*CAPL)
      PC = PC*(1. - X3/(2.*CAPL) * SQRT(CAPE*CAPI/ALPHA))
      RP = CAPF - PC
      RETURN
3     CONTINUE
      RP = X1 - X4
      RETURN
4     CONTINUE
      DEL = (4. * CAPF * CAPL**3)/(CAPE*X3**3 * X4)
      RP = DEL - .25
      RETURN
5     CONTINUE
      CAPM = CAPF * ( CAPL + X2 / 2. )
      CAPR = SQRT ( ( X2 * X2 ) / 4. + ( ( X3 + X1 ) / 2. )**2 )
      CAPJ = 2.*(.707*X1*X2*((X2*X2/12.) + ((X3 + X1)/2.) **2))
      TAUP = CAPF / (SQRT(2.) * X1 * X2 )
      TAUPP = CAPM * CAPR / CAPJ
      TAU = SQRT(TAUP*TAUP+2.* TAUP*TAUPP*(X2/(2.*CAPR))+TAUPP*TAUPP)
      RP = TAU - 13600.31
      RETURN
1000  CONTINUE
      RP = 1.10471 * X1*X1*X2 + .6735*X3*X4 + .04811 * X2*X3*X4
      RETURN
      END
```

Problem: C-16

```
      REAL FUNCTION STEEL(X,K)
      COMMON/PRINT/IBRKT,IPOW,IMS,JMSP,IIN,IOUT,IOTT
      DIMENSION X(20)
      LOGICAL FLAG
      DATA FLAG/.FALSE./
      IF(FLAG) GO TO 500
      PRINT(IOUT,*)" US STEEL, HIMMELBLAU NO. 22"
      FLAG = .TRUE.
500    CONTINUE
      IF(K .EQ. 0) GO TO 1000
      GO TO (1,2,3,4),K
1     CONTINUE
C     _ CONSTRAINTS HERE
      STEEL = 32.97 - 17.1 * X(1) - 38.2 * X(2) - 204.2 * X(3) -
     1  212.3 * X(4) - 623.4 * X(5) - 1495.5 * X(6) +
     2  169. * X(1) * X(3) + 3580. * X(3) * X(5) +3810. * X(4) *
     3  X(5) + 18500. * X(4) * X(6) + 24300.* X(5) * X(6)
      RETURN
2     CONTINUE
      STEEL = 25.12 - 17.9 * X(1) - 36.8 * X(2) - 113.9 * X(3) -
     1  169.7 * X(4) - 337.8 * X(5) - 1385.2 * X(6) + 139. * X(1) *
     2  X(3) + 2450. * X(4) * X(5) + 16600.* X(4) * X(6) +
     3  17200. * X(5) * X(6)
      RETURN
3     CONTINUE
      STEEL = -124.08 + 273.* X(2) + 70. * X(4) + 819.* X(5) -
     1  26000.* X(4) * X(5)
      RETURN
4     CONTINUE
      STEEL = -173.02 - 159.9 * X(1) + 311. * X(2) - 587.* X(4) -
     1  391.* X(5) - 2198.* X(6) + 14000.* X(1) * X(6)
      RETURN
1000  CONTINUE
      STEEL = 4.3 * X(1) + 31.8 * X(2) + 63.3 * X(3) + 15.8 * X(4) +
     1  68.5 * X(5) + 4.7 * X(6)
      RETURN
      END
```

Problem: C-17

```
      REAL FUNCTION CH3(X,K)
      COMMON/PRINT/IBRKT,IPOW,IMS,JMSP,IIN,IOUT,IOTT
      DIMENSION X(20)
      DIMENSION A(77)
      LOGICAL FLAG
      DATA A/ .2 , .31 , .4 , .44 , .6 , .62 , .79 , .8 , .88 , .93 ,
     1  1. , 1.2 , 1.24 , 1.32 , 1.4 , 1.55 , 1.58 , 1.6 , 1.76 ,
     2  1.8 , 1.86 , 2. , 2.17 , 2.2 , 2.37 , 2.4 , 2.48 , 2.6 ,
     3  2.64 , 2.79 , 2.8 , 3. , 3.08 , 3.1 , 3.16 , 3.41 ,
     4  3.52 , 3.6 , 3.72 , 3.95 , 3.96 , 4. , 4.03 , 4.2 ,
     5  4.34 , 4.4 , 4.65 , 4.74 , 4.8 , 4.84 , 5. , 5.28 ,
     6  5.4 , 5.53 , 5.72 , 6. , 6.16 , 6.32 , 6.6 , 7. , 7.11 ,
     7  7.2 , 7.8 , 7.9 , 8. , 8.4 , 8.69 , 9. , 9.48 , 10.27 ,
     8  11. , 11.06 , 11.85 , 12. , 13. , 14. , 15. /
      DATA FLAG/.FALSE./
      IF(FLAG) GO TO 500
      PRINT(IOUT,*)" CH3  REINFORCED CONCRETE BEAM"
      FLAG = .TRUE.
500    CONTINUE
      X1 = A(IFIX(X(1) + .1))
      X2 = X(2) / 2.
      IF(K .EQ. 0) GO TO 1000
1     CONTINUE
C     _ CONSTRAINTS HERE
      CH3 = -(X1 - .2458 * X1**2 / X2 - 6.)
      RETURN
1000  CONTINUE
      CH3 = 29.4 * X1 + 18. * X2
      RETURN
      END
```

Problem: C-18

```
      REAL FUNCTION CH3B(X,K)
      COMMON/PRINT/IBRKT,IPOW,IMS,JMSP,IIN,IOUT,IOTT
      DIMENSION X(20)
      DIMENSION A(77)
      LOGICAL FLAG
      DATA A/ .2 , .31 , .4 , .44 , .6 , .62 , .79 , .8 , .88 , .93 ,
     1  1. , 1.2 , 1.24 , 1.32 , 1.4 , 1.55 , 1.58 , 1.6 , 1.76 ,
     2  1.8 , 1.86 , 2. , 2.17 , 2.2 , 2.37 , 2.4 , 2.48 , 2.6 ,
     3  2.64 , 2.79 , 2.8 , 3. , 3.08 , 3.1 , 3.16 , 3.41 ,
     4  3.52 , 3.6 , 3.72 , 3.95 , 3.96 , 4. , 4.03 , 4.2 ,
     5  4.34 , 4.4 , 4.65 , 4.74 , 4.8 , 4.84 , 5. , 5.28 ,
     6  5.4 , 5.53 , 5.72 , 6. , 6.16 , 6.32 , 6.6 , 7. , 7.11 ,
     7  7.2 , 7.8 , 7.9 , 8. , 8.4 , 8.69 , 9. , 9.48 , 10.27 ,
     8  11. , 11.06 , 11.85 , 12. , 13. , 14. , 15. /
      DATA FLAG/.FALSE./
      IF(FLAG) GO TO 500
      PRINT(IOUT,*)" CH3B  REINFORCED CONCRETE BEAM"
      FLAG = .TRUE.
500    CONTINUE
      X1 = A(IFIX(X(1) + .1))
      X2 = X(2) / 2.
      IF(K .EQ. 0) GO TO 1000
1     CONTINUE
C     _ CONSTRAINTS HERE
      CH3B = -(X1 - .2458 * X1**2 / X2 - 6.)
      RETURN
1000  CONTINUE
      CH3B = 44.4 * X1 + 18. * X2
      RETURN
      END
```

Problem: C-19

```
      REAL FUNCTION GM1(X,K)
      COMMON/PRINT/IBRKT,IPOW,IMS,JMSP,IIN,IOUT,IOTT
      DIMENSION X(20)
      DIMENSION DH(4)
      LOGICAL FLAG
      DATA FLAG/.FALSE./
      DATA DH/15. , 25. , 40. , 60./
      IF(FLAG) GO TO 500
      PRINT(IOUT,*)" HATCH COVER"
      FLAG = .TRUE.
500    CONTINUE
      TF = X(1)/10.
      H = DH(IFIX(X(2) + .1))
      IF(K .EQ. 0) GO TO 1000
      GO TO (1,2,3,4),K
1     CONTINUE
      GM1 = 1800./H - 450.
      RETURN
2     CONTINUE
      GM1 = 4500./ (TF * H) - 700.
      RETURN
3     CONTINUE
      GM1 = 4500./(TF * H) - 700. * TF**2
      RETURN
4     CONTINUE
      GM1 = 5.62 / (7. * TF * H **2 ) - .0025
      RETURN
1000  CONTINUE
      GM1 = H + 120. * TF
      RETURN
      END
```

Problem: C-20

```
      REAL FUNCTION STEAM(X,K)
      COMMON/PRINT/IBRKT,IPOW,IMS,JMSP,IIN,IOUT,IOTT
      DIMENSION X(20)
      DIMENSION DT(6) , WT(6) , GMUT(23)
      LOGICAL FLAG
      DATA FLAG/.FALSE./
      IF(FLAG) GO TO 500
      DATA GMUT/3.73 , 3.17 , 2.71 , 2.36 , 2.08 , 1.85 , 1.66 ,
     1 1.49 , 1.36 , 1.24 , 1.14 , 1.04 , .97 , .9 , .84 ,
     2 .786 , .738 , .695 , .654 , .618 , .585 , .555 , .528/
      DATA DT/.625 , .75 , 1. , 1.25 , 1.5 , 2./
      DATA WT/ .134 , .109 , .083 , .065 , .049 , .035/
      PRINT(IOUT,*)" STEAM CONDENSER"
      SPA = 2.
      SP = SPA - 14.7
      TSAT = 126.08
      HFG = 1022.2
      TWI = 70.
      M = 5380
      PI = 3.14159
      RSK = 160.
      RF = .001
      HO = 2290.
      FLAG = .TRUE.
500    CONTINUE
      D = DT(IFIX(X(1) + .1))
      W = WT(IFIX(X(2) + .1))
      L = X(3)*4.
      N = X(4)
      SN = X(5)
      SW = X(6)
      SW = X(6)
      RI = (D/2. - W) / 12.
      SWP = SW * 62.4 * 60. / 7.48
      DELT = (HFG * M) /SWP
      RO = D/24.
      IF(K .EQ. 0) GO TO 1000
      GO TO (1,2,3,4,5),K
1     CONTINUE
C     _ CONSTRAINTS HERE
      STEAM = 2.*W - D
      RETURN
2     CONTINUE
      TWO = TWI + DELT
      STEAM = TWO - TSAT
      RETURN
3     CONTINUE
      V = SW / (7.481 * 60. * PI * RI**2 * SN)
      TBAR = TWI + DELT/2.
      K1 = (TBAR/10.) - 3
      K2 = K1 + 1
      R = TBAR/10. - (3. + FLOAT(K1))
      GMU = GMUT(K1) + R * (GMUT(K2)- GMUT(K1))
      RE = 300. * 62.4 * 2. * RI * V / GMU
      STEAM = 3000. - RE
      RETURN
4     CONTINUE
      V = SW / (7.481 * 60. * PI * RI**2 * SN)
      TBAR = TWI + DELT/2.
      K1 = (TBAR/10.) - 3
      K2 = K1 + 1
      R = TBAR/10. - (3. + FLOAT(K1))
      GMU = GMUT(K1) + R * (GMUT(K2)- GMUT(K1))
      RE = 300. * 62.4 * 2. * RI * V / GMU
      TWO = TWI + DELT
      TBAR = TWI + DELT/2.
      DELTMX = TSAT - TWI
      DELTMN = TSAT - TWO
      IF(DELTMN .LE. 1.) DELTMN = 1.
      THMIN = (DELTMX - DELTMN)/ALOG(DELTMX/DELTMN)
      A0 = PI * D * L * SN * N / 12.
      HI = (150. * (1. + .011*TBAR) * V**.8)/(24. * RI)**.2
      U = 1./((RO/RI)*1./HI + RF) + ((RO/RSK) * ALOG(RO/RI))
     1  + 1./HO
      Q0 = U * A0 * THMIN
      STEAM = (HFG * M) - Q0
      RETURN
5     CONTINUE
      TBAR = TWI + DELT/2.
      K1 = (TBAR/10.) - 3
      K2 = K1 + 1
      R = TBAR/10. - (3. + FLOAT(K1))
      GMU = GMUT(K1) + R * (GMUT(K2)- GMUT(K1))
      RE = 300. * 62.4 * 2. * RI * V / GMU
      V = SW / (7.481 * 60. * PI * RI**2 * SN)
      SF = .0014 + .125 / RE**.32
      DELP = (SF * V**2 * L) / (32.2 * RI)
      DELPE = V**2 / 64.4
      DELPC = V**2 / 128.8
      H =(DELP + DELPE + DELPC) * N
      F = 62.4 * H / 144. - SP
      WM = (F*D) / (8000. + .8*F)
      STEAM = WM - W
      RETURN
1000  CONTINUE
      DS = SQRT(3. * D**2 * SN * N / 144.)
      TS = (ABS(SP) * DS) / 32.E3 + .0104
      TE = (ABS(SP) * DS) / 64.E3 + .0104
      WS = (PI*DS * L * TS + PI * DS**2 * TE) * 489.
      CS = 1.1 * WS
      CT = 1.5 * L * PI * SN * N * (D**2 - (D - 2*W)**2) * .322 / 4.
      HP = .19 + .00045 * SW * H
      CPM = 820. * HP **.467
      STEAM = CS + CT + CPM
      RETURN
      END
```

Problem: C-21

```
      REAL FUNCTION CH1(X,K)
      COMMON/PRINT/IBRKT,IPOW,IMS,JMSP,IIN,IOUT,IOTT
      DIMENSION X(20)
      LOGICAL FLAG
      DATA FLAG/.FALSE./
      IF(FLAG) GO TO 500
      PRINT(IOUT,*) "CH1, TIMBER FRAME"
      FLAG = .TRUE.
500   CONTINUE
      IF(K .EQ. 0) GO TO 1000
      X12 = X(1) **2
      X22 = X(2) **2
      X23 = X22 * X(2)
      X13 = X12 * X(1)
      XXX = X23 / X13
      F1 = 5832. / (12. + 5.33 * XXX)
      F2 = 4.5 / ((8. + 3.56 * XXX) * X(2))
      GO TO (1,2,3),K
1     CONTINUE
C      CONSTRAINTS HERE
      CH1 = -( 1.8 - 2.25/X(1) - F1/X12)
      RETURN
2     CONTINUE
      CH1 = -(1.8 - F2 - F1.X22)
      RETURN
3     CONTINUE
      CH1 = -(1.8 - F2 - (729. - F1)/X22)
      RETURN
1000  CONTINUE
      CH1 = 1152. * X(1) + 864. * X(2)
      RETURN
      END
```

Problem: U-1

```
      REAL FUNCTION AG1(X,K)
      COMMON/PRINT/IBRKT,IPOW,IMS,JMSP,IIN,IOUT,IOTT
      DIMENSION X(20)
      LOGICAL FLAG
      DATA FLAG/.FALSE./
      IF(FLAG) GO TO 500
      PRINT(IOUT,*)"AG 1, KUESTER & MIZE"
      FLAG = .TRUE.
500    CONTINUE
      IF(K .EQ. 0) GO TO 1000
1     CONTINUE
1000  CONTINUE
      AG1 = -(3803.84 + 138 * X(1) + 239.92 * X(2) - 123.08 * X(1)**2 -
     1  203.64 * X(2)**2 - 182.25 * X(1) * X(2) )
      RETURN
      END
```

Problem: U-2

```
      REAL FUNCTION AG2(X,K)
      COMMON/PRINT/IBRKT,IPOW,IMS,JMSP,IIN,IOUT,IOTT
      DIMENSION X(20)
      LOGICAL FLAG
      DATA FLAG/.FALSE./
      IF(FLAG) GO TO 500
      PRINT(IOUT,*)"AG 2, HIMMELBLAU 28"
      FLAG = .TRUE.
500    CONTINUE
      IF(K .EQ. 0) GO TO 1000
1     CONTINUE
1000  CONTINUE
      AG2 = (X(1)**2 + X(2) - 11.)**2 + (X(1) + X(2)**2 - 7.)**2
      RETURN
      END
```

Problem: U-3

```
      REAL FUNCTION AG3(X,K)
      COMMON/PRINT/IBRKT,IPOW,IMS,JMSP,IIN,IOUT,IOTT
      DIMENSION X(20)
      LOGICAL FLAG
      DATA FLAG/.FALSE./
      IF(FLAG) GO TO 500
      PRINT(IOUT,*)" AG3, AG2 SCALED"
      FLAG = .TRUE.
500    CONTINUE
```

```
      IF(K .EQ. 0) GO TO 1000
1     CONTINUE
1000  CONTINUE
      AG3 = (9. * X(1)**2 + 2. * X(2) - 11.)**2 +
     1  (3. * X(1) + 4. * X(2)**2 - 7.)**2
      RETURN
      END
```

Problem: U-4

```
      REAL FUNCTION AG4(X,K)
      COMMON/PRINT/IBRKT,IPOW,IMS,JMSP,IIN,IOUT,IOTT
      DIMENSION X(20)
      LOGICAL FLAG
      DATA FLAG/.FALSE./
      IF(FLAG) GO TO 500
      PRINT(IOUT,*)"AG 4, HIMMELBLAU 19, ROSENBROCKS"
      FLAG = .TRUE.
500    CONTINUE
      IF(K .EQ. 0) GO TO 1000
1     CONTINUE
1000  CONTINUE
      AG4 = 100. * (X(2) - X(1)**2)**2 + (1. - X(1))**2
      RETURN
      END
```

Problem: U-5

```
      REAL FUNCTION AG5(X,K)
      COMMON/PRINT/IBRKT,IPOW,IMS,JMSP,IIN,IOUT,IOTT
      DIMENSION X(20)
      LOGICAL FLAG
      DATA FLAG/.FALSE./
      IF(FLAG) GO TO 500
      PRINT(IOUT,*)" GLANKWAHMDEE NO. 5,  HIMMELBLAU NO. 31"
      FLAG = .TRUE.
500    CONTINUE
      IF(K .EQ. 0) GO TO 1000
1     CONTINUE
1000  CONTINUE
      F1 = (X(1) - 2.)**2 + (X(2) - 1.)**2
      G1 = X(1)**2/(-4.) - X(2)**2 + 1.
      IF(ABS(G1) .LT. 1.E-6) G1 = SIGN(1.E-6,G1)
      H1 = X(1) - 2. * X(2) + 1.
      AG5 = F1 + .04/G1 + H1**2/.2
      RETURN
      END
```

Problem: U-6

```
      REAL FUNCTION AG6(X,K)
      COMMON/PRINT/IBRKT,IPOW,IMS,JMSP,IIN,IOUT,IOTT
      DIMENSION X(20)
      LOGICAL FLAG
      DATA FLAG/.FALSE./
      IF(FLAG) GO TO 500
      PRINT(IOUT,*)"AG # 6, HIMMELBLAU #26"
      FLAG = .TRUE.
500    CONTINUE
      IF(K .EQ. 0) GO TO 1000
1     CONTINUE
1000  CONTINUE
      AG6=(X(1) + 10.*X(2))**2 + 5.*(X(3) - X(4))**2
      AG6 = AG6 + (X(2) - 2.*X(3))**4 + 10.*(X(1) - X(4))**4
      RETURN
      END
```

Problem: U-7

```
      REAL FUNCTION AG7(X,K)
      COMMON/PRINT/IBRKT,IPOW,IMS,JMSP,IIN,IOUT,IOTT
      DIMENSION X(20)
      LOGICAL FLAG
      DATA FLAG/.FALSE./
      IF(FLAG) GO TO 500
      PRINT(IOUT,*)" AG 7"
      FLAG = .TRUE.
C      ONE TIME CALCULATIONS HERE
500    CONTINUE
C      EVERYTIME CALCULATIONS HERE
      IF(K .EQ. 0) GO TO 1000
1     CONTINUE
1000  CONTINUE
      AG7 = 100. * (X(2) - X(1)**2)**2 + (1. - X(1))**2 +
     1  90. * (X(4) - X(3)**2)**2 + (1. - X(3))**2 +
     2  10.1 * ((X(2) - 1.)**2 + (X(4) - 1.)**2) +
     3  19.8 * (X(2) - 1.) * (X(4) - 1.)
      RETURN
      END
```

Problem: U-8

```
      REAL FUNCTION AG8(X,K)
      COMMON/PRINT/IBRKT,IPOW,IMS,JMSP,IIN,IOUT,IOTT
      DIMENSION X(20)
      DIMENSION C(5),D(5,5),W(5)
      LOGICAL FLAG
      DATA FLAG/.FALSE./
      DATA C/-15. , -27. , -36. , -18. , -12./
      DATA (D(1,I),I=1,5)/35. , -20. , -10. , 32. , -10./
      DATA (D(2,I),I=1,5)/-20. , 40. , -6. , -31. , 32./
      DATA (D(3,I),I=1,5)/-10. , -6. , 11. , -6. , -10./
      DATA (D(4,I),I=1,5)/32. , -31. , -6. , 38. , -20./
      DATA(D(5,I),I=1,5)/-10. , 32. , -10. , -20. , 31./
      IF(FLAG) GO TO 500
      PRINT(IOUT,*)" GLANKWAHMDEE NO. 8"
      FLAG = .TRUE.
C      ONE TIME CALCULATIONS HERE
500    CONTINUE
C      EVERYTIME CALCULATIONS HERE
      IF(K .EQ. 0) GO TO 1000
1     CONTINUE
1000  CONTINUE
      DO 10 I = 1,5
      W(I) = 0.
      DO 5 J = 1,5
      W(I) = W(I) + D(I,J) * X(J)
5     CONTINUE
10    CONTINUE
      F = 0.
      DO 20 I = 1,5
      F = F + (C(I) + W(I)) * X(I)
20    CONTINUE
      AG8 = F
      RETURN
      END
```

Problem: U-9

```
      REAL FUNCTION AMMO(X,K)
      COMMON/PRINT/IBRKT,IPOW,IMS,JMSP,IIN,IOUT,IOTT
      DIMENSION X(20)
      LOGICAL FLAG
      DATA FLAG/.FALSE./
      IF(FLAG) GO TO 500
      PRINT(IOUT,*)" STOECKER:  AMMONIA STORAGE TANK"
      FLAG = .TRUE.
C      ONE TIME CALCULATIONS HERE
500    CONTINUE
C      EVERYTIME CALCULATIONS HERE
      IF(K .EQ. 0) GO TO 1000
1     CONTINUE
1000  CONTINUE
      AMMO = 400. * X(1)**.9 + 1000. +
     1  22. * (EXP((-3950. / (X(2)+460.)) + 11.86)-14.7) **1.2
     1  + 144. * (80. - X(2)) / X(1)
      RETURN
      END
```

Problem: U-10

```
      REAL FUNCTION DUCT1(X,K)
      COMMON/PRINT/IBRKT,IPOW,IMS,JMSP,IIN,IOUT,IOTT
      DIMENSION X(20)
      LOGICAL FLAG
      DATA FLAG/.FALSE./
      IF(FLAG) GO TO 500
      PRINT(IOUT,*)" DUCT LAYOUT 1"
      FLAG = .TRUE.
C      ONE TIME CALCULATIONS HERE
      R12 = 30.
      R23 = 20.
      R36 = 65.
      R34 = 25.
      R27 = 20.
      R78 = 20.
      R710 = 35.
      RO = 2.
      F = .017
      ETA = .8
      Q12 = 6500.
      Q23 = 3000.
      Q27 = 3500.
      Q36 = 2000.
      Q34 = 1000.
      Q78 = 2000.
      Q710 = 1500.
500    CONTINUE
C      EVERYTIME CALCULATIONS HERE
      X12 = X(1) / 12.
      X23 = X(2) / 12.
      X27 = X(3) / 12.
      IF(K .EQ. 0) GO TO 1000
1     CONTINUE
1000  CONTINUE
      DUCT1 = 1.E9
C     UPSTREAM PRESSURE AT 2
      RD12=  RDUCT(RO,F,R12,X12)
      PU2 = .85 - RD12*(Q12/4000.)**2
C     BRANCH AND DOWNSTREAM PRESSURES AT 2
      BLAM2 = BLAM(Q23,Q12,X23,X12)
      RB2 = RUTOB1(RO,BLAM2,Q23,Q12,X23,X12)
      PB2 = PU2 - RB2*(Q23/4000.)**2
      RD2= RUTOD(ETA,RO,Q27,Q12,X27,X12)
      PD2 = PU2 - RD2*(Q27/4000.)**2
C     UPSTREAM PRESSURE AT 3
      RD23= RDUCT(RO,F,R23,X23)
      PU3 = PB2 - RD23*(Q23/4000.)**2
C     UPSTREAM PRESSURE AT 7
      RD27= RDUCT(RO,F,R27,X27)
      PU7 = PD2 - RD27*(Q27/4000.)**2
C     DETERMINE SMALLEST ACCEPTABLE SIZES FOR REMAINING FITTINGS
C     OUTLET 6
      DO 1010 ID = 6,24
      X36 = FLOAT(ID) / 12.
      RD3 = RUTOD(ETA,RO,Q36,Q23,X36,X23)
      PD3 = PU3 - RD3*(Q36/4000.)**2
      EL = 12.19*X36 + R36
      RD36= RDUCT(RO,F,EL,X36)
      P6 = PD3 - RD36*(Q36/4000.)**2
      IF(P6 .GE. .1) GO TO 1020
1010  CONTINUE
C     INSUFFICIENT DUCT SIZE
      DUCT1 = DUCT1- P6
1020  DO 1030 ID = 6,24
      X34 = FLOAT(ID) / 12.
      BLAM3 = BLAM(Q34,Q23,X34,X23)
      RB3 = RUTOB1(RO,BLAM3,Q34,Q23,X34,X23)
      PB3 = PU3 - RB3*(Q34/4000.)**2
      RD34= RDUCT(RO,F,R34,X34)
      P4 = PB3 - RD34*(Q34/4000.)**2
      IF(P4 .GE. .1) GO TO 1040
1030  CONTINUE
      DUCT1 = DUCT1- P4
1040  DO 1050 ID = 6,24
      X78 = FLOAT(ID) / 12.
      RD7 = RUTOD(ETA,RO,Q78,Q27,X78,X27)
      PD7 = PU7 - RD7*(Q78/4000.)**2
      RD78= RDUCT(RO,F,R78,X78)
      P8 = PD7 - RD78*(Q78/4000.)**2
      IF(P8 .GE. .1) GO TO 1060
1050  CONTINUE
      DUCT1 = DUCT1- P8
1060  DO 1070 ID = 6,24
      X710 = FLOAT(ID) / 12.
      BLAM7 = BLAM(Q710,Q27,X710,X27)
      RB7 = RUTOB1(RO,BLAM7,Q710,Q27,X710,X27)
      PB7 = PU7 - RB7*(Q710/4000.)**2
      EL = 12.19 * X710 + R710
      RD710= RDUCT(RO,F,EL,X710)
      P10 = PB7 - RD710*(Q710/4000.)**2
      IF(P10 .GE. .1) GO TO 1080
1070  CONTINUE
      DUCT1 = DUCT1- P10
1080  CONTINUE
      X(4) = X36 * 12.
      X(5) = X34 * 12.
      X(6) = X78 * 12.
      X(7) = X710 * 12.
      IF(DUCT1 .NE. 1.E9) RETURN
      DUCT1 = COST(X12,R12) + COST(X23,R23) + COST(X36,R36) +
     1  COST(X34,R34) + COST(X27,R27) + COST(X78,R78) +
     2  COST(X710,R710)
      RETURN
      END

      REAL FUNCTION COST(X,L)
      REAL L
      DATA PI/3.14159/
      F = 1.2 * .906
      IF(X .GT. 1.125) F = 1.2 * 1.156
      IF(X .GT. 1.875) F = 1.2 * 1.406
      IF(X .GT. 3.04) F = 1.2 * 1.656
      IF(X .GT. 4.21) F = 1.2 * 2.156
      IF(X .GT. 5.04) F = 1.2 * 2.656
      F = F * 1.35
      COST = PI * X * L * F
      RETURN
      END

      REAL FUNCTION BLAM(QB,QU,DB,DU)
      BLAM = .51*(VEL(QB,DB)/VEL(QU,DU))**2 + 1.
      RETURN
      END

      REAL FUNCTION VEL(Q,D)
      VEL = Q/AF(D)
      RETURN
      END

      FUNCTION RDUCT(RO,F,L,D)
C
C     RESISTANCE FOR STRAIGHT DUCT ELEMENT
C     RO=AIR DENSITY, KG/M**3
C     F=FRICTION FACTOR
C     L=LENGTH OF DUCT,M
C     DIAMETER OF DUCT,M
C
C***********************************
C
C    THIS AND THE FOLLOWING FUNCTIONS CALCULATE THE RESISTANCE
C    TERM FOR DUCT ELEMENTS WHICH CORRESPONDS TO THE EQUATION:
C
C           P1-P2= R*Q**2
C
C    WHERE P1 AND P2 ARE THE STATIC PRESSURES AT 1 AND 2,
C    AND Q IS THE VOLUME FLOW RATE.
C
```

```
C    THE FUNCTIONS ARE SET UP FOR THE SI UNIT SYSTEM
C    WITH  Q IN M**3/S   D IN M,  P IN PASCALS,AND RO IN KG/M**3
C
C
C    TO USE IN THE ENGLISH SYSTEM, CALL THE FUNCTIONS
C    WITH (Q/4000) HAVING Q IN CUBIC FT PER MIN,  RO=RO/RO(STD)
C     WHERE RO(STD) IS THE DENSITY AT 60 F  (.075 LBM/CU FT),
C     P IN INCHES OF WATER COLUMN, AND D IN FEET
C****************************************
C
C
      REAL L
      A=AF(D)
      RDUCT=RO*F*L/(2.*D*A**2)
      RETURN
      END

       FUNCTION RUTOB1(RO,CU,QB,QU,DB,DU)
C
C     UPSTREAM TO BRANCH RESISTANCE USING LOSS COEFFICIENT BASED ON
C     UPSTREAM VELOCITY
C     RO=AIR DENSITY  KG/M**3
C     CU=LOSS COEFFICIENT
C     QB & QU  VOLUME FLOW IN BRANCH AND UPSTREAM, M**3/S
C     DB &DU  BRANCH AND UPSTREAM DIAMETERS, M
C
      AB=AF(DB)
      AU=AF(DU)
      VBOVU=QB*AU/(QU*AB)
      VBOVUS=(VBOVU)**2
      RUTOB1=RO/(2.*AB**2)*(VBOVUS-1.+CU)/VBOVUS
      RETURN
      END

      FUNCTION RUTOD(ETA,RO,QD,QU,DD,DU)
C
C     UPSTREAM TO DOWNSTREAM RESISTANCE WITH REGAIN EFFICIENCY
C     OF ETA.
C     RO=AIR DENSITY  KG/M**3
C     QU &QD  UPSTREAM AND DOWNSTREAM VOLUME FLOW RATES, M**3/S
C     DU & DD UPSTREAM AND DOWNSTREAM DIAMETERS, M
C
       AD=AF(DD)
       AU=AF(DU)
       VUOVD=QU*AD/(QD*AU)
      RUTOD=ETA*RO/(2.*AD**2)*(1.-(VUOVD)**2)
      RETURN
      END

       FUNCTION AF(D)
C
C     AREA CALCULATION FOR CURCULAR DUCT OF DIAMETER D
```

```
C
        PI=3.14159
        AF=PI*D**2/4.
        RETURN
        END
```

Problem: U-11

```
      REAL FUNCTION DUCT4(X,K)
      COMMON/PRINT/IBRKT,IPOW,IMS,JMSP,IIN,IOUT,IOTT
      DIMENSION X(20)
      LOGICAL FLAG
      DATA FLAG/.FALSE./
      IF(FLAG) GO TO 500
      PRINT(IOUT,*)" DUCT LAYOUT 4"
      FLAG = .TRUE.
C       ONE TIME CALCULATIONS HERE
      R12 = 30.
      R23 = 20.
      R36 = 65.
      R34 = 25.
      R27 = 20.
      R78 = 20.
      R710 = 35.
      RO = 2.
      F = .017
      ETA = .8
      Q12 = 6500.
      Q23 = 3000.
      Q27 = 3500.
      Q36 = 2000.
      Q34 = 1000.
      Q78 = 2000.
      Q710 = 1500.
500     CONTINUE
C       EVERYTIME CALCULATIONS HERE
      X12 = X(1) / 12.
      X23 = X(2) / 12.
      X27 = X(3) / 12.
      IF(K .EQ. 0) GO TO 1000
1     CONTINUE
1000  CONTINUE
      DUCT4 = 1.E9
C     UPSTREAM PRESSURE AT 2
      RD12=  RDUCT(RO,F,R12,X12)
      PU2 = .258 - RD12*(Q12/4000.)**2
C     BRANCH AND DOWNSTREAM PRESSURES AT 2
      BLAM2 = BLAM(Q23,Q12,X23,X12)
      RB2 = RUTOB1(RO,BLAM2,Q23,Q12,X23,X12)
      PB2 = PU2 - RB2*(Q23/4000.)**2
      RD2= RUTOD(ETA,RO,Q27,Q12,X27,X12)
      PD2 = PU2 - RD2*(Q27/4000.)**2
C     UPSTREAM PRESSURE AT 3
      RD23= RDUCT(RO,F,R23,X23)
      PU3 = PB2 - RD23*(Q23/4000.)**2
C     UPSTREAM PRESSURE AT 7
      RD27= RDUCT(RO,F,R27,X27)
      PU7 = PD2 - RD27*(Q27/4000.)**2
C     DETERMINE SMALLEST ACCEPTABLE SIZES FOR REMAINING FITTINGS
C     OUTLET 6
      DO 1010 ID = 6,35
      X36 = FLOAT(ID) / 12.
      RD3 = RUTOD(ETA,RO,Q36,Q23,X36,X23)
      PD3 = PU3 - RD3*(Q36/4000.)**2
      EL = 12.19*X36 + R36
      RD36= RDUCT(RO,F,EL,X36)
      P6 = PD3 - RD36*(Q36/4000.)**2
      IF(P6 .GE. .1) GO TO 1020
1010  CONTINUE
C     INSUFFICIENT DUCT SIZE
      DUCT4 = DUCT4- P6
1020  DO 1030 ID = 6,35
      X34 = FLOAT(ID) / 12.
      BLAM3 = BLAM(Q34,Q23,X34,X23)
      RB3 = RUTOB1(RO,BLAM3,Q34,Q23,X34,X23)
      PB3 = PU3 - RB3*(Q34/4000.)**2
      RD34= RDUCT(RO,F,R34,X34)
      P4 = PB3 - RD34*(Q34/4000.)**2
      IF(P4 .GE. .1) GO TO 1040
1030  CONTINUE
      DUCT4 = DUCT4- P4
1040  DO 1050 ID = 6,35
      X78 = FLOAT(ID) / 12.
      RD7 = RUTOD(ETA,RO,Q78,Q27,X78,X27)
      PD7 = PU7 - RD7*(Q78/4000.)**2
      RD78= RDUCT(RO,F,R78,X78)
      P8 = PD7 - RD78*(Q78/4000.)**2
      IF(P8 .GE. .1) GO TO 1060
1050  CONTINUE
      DUCT4 = DUCT4- P8
1060  DO 1070 ID = 6,35
      X710 = FLOAT(ID) / 12.
      BLAM7 = BLAM(Q710,Q27,X710,X27)
      RB7 = RUTOB1(RO,BLAM7,Q710,Q27,X710,X27)
      PB7 = PU7 - RB7*(Q710/4000.)**2
      EL = 12.19 * X710 + R710
      RD710= RDUCT(RO,F,EL,X710)
      P10 = PB7 - RD710*(Q710/4000.)**2
      IF(P10 .GE. .1) GO TO 1080
1070  CONTINUE
      DUCT4 = DUCT4- P10
1080  CONTINUE
```

```
      X(4) = X36 * 12.
      X(5) = X34 * 12.
      X(6) = X78 * 12.
      X(7) = X710 * 12.
      IF(DUCT4 .NE. 1.E9) RETURN
      DUCT4 = COST(X12,R12) + COST(X23,R23) + COST(X36,R36) +
     1  COST(X34,R34) + COST(X27,R27) + COST(X78,R78) +
     2  COST(X710,R710)
      RETURN
      END
```

See problem U-10 for subroutines RDUCT, RUTOB1, RUTOD, AF, COST, BLAM, and VEL.

Problem: U-12

```
      REAL FUNCTION DUCT9(X,K)
      COMMON/PRINT/IBRKT,IPOW,IMS,JMSP,IIN,IOUT,IOTT
      DIMENSION X(20)
      LOGICAL FLAG
      DATA FLAG/.FALSE./
      IF(FLAG) GO TO 500
      PRINT(IOUT,*)" DUCT LAYOUT 9  .264 IN. H2O TOTAL FAN PRESSURE"
      FLAG = .TRUE.
C      ONE TIME CALCULATIONS HERE
      RO = 2.
      F = .017
      ETA = .8
      R12 = 20.
      R23 = 10.
      R28 = 15.
      R36 = 10.
      R35 = 30.
      Q12 = 2500.
      Q23 = 1750.
      Q28 = 750.
      Q36 = 750.
      Q35 = 1000.
500    CONTINUE
C      EVERYTIME CALCULATIONS HERE
      X12 = X(1) / 12.
      X23 = X(2) / 12.
      IF(K .EQ. 0) GO TO 1000
1     CONTINUE
1000  CONTINUE
      DUCT9 = 1.E9
      RD12 = RDUCT(RO,F,R12,X12)
      PU2 = .264 - RD12 * (Q12 / 4000.) **2
      RD2 = RUTOD(ETA,RO,Q23,Q12,X23,X12)
      PD2 = PU2 - RD2 * (Q23 / 4000.) **2
      RD23 = RDUCT(RO,F,R23,X23)
      PU3 = PD2 - RD23 * (Q23 / 4000.) **2
      DO 1010 ID = 6,35
      X28 = FLOAT(ID) / 12.
      BLAM2 = BLAM(Q28,Q12,X28,X12)
      RB2 = RUTOB1(RO,BLAM2,Q28,Q12,X28,X12)
      PB2 = PU2 - RB2 * (Q28 / 4000.) **2
      EL = 12.19 * X28 + R28
      RD28 = RDUCT(RO,F,EL,X28)
      P8 = PB2 - RD28 * (Q28 / 4000.) **2
      IF(P8 .GE. .12) GO TO 1020
1010  CONTINUE
      DUCT9 = DUCT9 - P8
1020  DO 1030 ID = 6,35
      X36 = FLOAT(ID) / 12.
      BLAM3 = BLAM(Q36,Q23,X36,X23)
      RB3 = RUTOB1(RO,BLAM3,Q36,Q23,X36,X23)
      PB3 = PU3 - RB3 * (Q36 / 4000.) **2
      RD36 = RDUCT(RO,F,R36,X36)
      P6 = PB3 - RD36 * (Q36 / 4000.) **2
      IF(P6 .GE. .12) GO TO 1040
1030  CONTINUE
      DUCT9 = DUCT9 - P6
1040  DO 1050 ID = 6,35
      X35 = FLOAT(ID) / 12.
      RD3 = RUTOD(ETA,RO,Q35,Q23,X35,X23)
      PD3 = PU3 - RD3 * (Q35 / 4000.) **2
      EL = 12.19 * X35 + R35
      RD35 = RDUCT(RO,F,EL,X35)
      P5 = PD3 - RD35 * (Q35 / 4000.) **2
      IF(P5 .GE. .12) GO TO 1060
1050  CONTINUE
      DUCT9 = DUCT9 - P5
1060  CONTINUE
      X(3) = X28 * 12.
      X(4) = X36 * 12.
      X(5) = X35 * 12.
      IF(DUCT9 .NE. 1.E9) RETURN
      DUCT9 = COST(X12,R12) + COST(X23,R23) + COST(X28,R28) +
     1  COST(X36, R36) + COST(X35, R35)
      RETURN
      END
```

See problem U-10 for subroutines RDUCT, RUTOB1, RUTOD, AF, COST, BLAM, and VEL.

Problem: U-13

```
      REAL FUNCTION GEAR1(X,K)
      COMMON/PRINT/IBRKT,IPOW,IMS,JMSP,IIN,IOUT,IOTT
      DIMENSION X(20)
      LOGICAL FLAG
      DATA FLAG/.FALSE./
      IF(FLAG) GO TO 500
      PRINT(IOUT,*)" EASON AND FENTON: MIN. INERTIA GEAR TRAIN"
      FLAG = .TRUE.
C      ONE TIME CALCULATIONS HERE
500    CONTINUE
      IF(K .EQ. 0) GO TO 1000
1     CONTINUE
1000  CONTINUE
      GEAR1 = .1* (12.+X(1)**2 + (1. + X(2)**2)/X(1)**2 + (X(1)**2
     1  *X(2)**2 + 100.) / (X(1) * X(2))**4 )
      RETURN
      END
```

Problem: U-14

```
      REAL FUNCTION WATER(X,K)
      COMMON/PRINT/IBRKT,IPOW,IMS,JMSP,IIN,IOUT,IOTT
      DIMENSION X(20)
      LOGICAL FLAG
      DATA FLAG/.FALSE./
      IF(FLAG) GO TO 500
      PRINT(IOUT,*)" ROSENBROCK AND STOREY HEAVY WATER PLANT"
      FLAG = .TRUE.
C      ONE TIME CALCULATIONS HERE
500    CONTINUE
      IF(K .EQ. 0) GO TO 1000
1     CONTINUE
1000  CONTINUE
      RN = X(1)
      R = X(2)
      T = X(3)
      ALP = EXP(508./T - .382)
      BET = R / 1400.
      PHI = ((ALP - 1.)/ALP)*(ALP*BET)**(RN+1.) + BET - 1.
      F = (PHI*(1.-BET))/(.6*(1.-BET)*(ALP*BET - 1.) + .4*PHI)
      H = 2. + 3. *EXP(16.875 - T/14.4)
      WATER = (300. * R + 4000. * RN * H + 80000.) / (18.3*(F-1.))
      RETURN
      END
```

Problem: U-15

```
      REAL FUNCTION POW2(X,K)
      COMMON/PRINT/IBRKT,IPOW,IMS,JMSP,IIN,IOUT,IOTT
      DIMENSION X(20)
      LOGICAL FLAG
      DATA FLAG/.FALSE./
      IF(FLAG) GO TO 500
      PRINT(IOUT,*)" FLETCHER AND POWELL,  HIMMELBLAU NO. 34"
      FLAG = .TRUE.
C      ONE TIME CALCULATIONS HERE
      PI = 3.141592654
500    CONTINUE
C      EVERYTIME CALCULATIONS HERE
      IF(K .EQ. 0) GO TO 1000
1     CONTINUE
1000  CONTINUE
      IF(ABS(X(2)) .GT. 1.E-6) GO TO 10
      TH = 0.
      GO TO 90
10    IF(ABS(X(1)) .GT. 1.E-6) GO TO 20
      TH = .25
      GO TO 90
20    TH = ATAN(X(2)/X(1))/(2.*PI)
90    CONTINUE
      IF(X(1) .LT. 0) TH = TH + .5
      POW2 = 100. * ((X(3) - 10. * TH)**2 + (SQRT(X(1)**2 +
     1  X(2)**2) - 1.)**2) + X(3)**2
      RETURN
      END
```

Problem: U-16

```
      REAL FUNCTION OBJT3(X,K)
      COMMON/PRINT/IBRKT,IPOW,IMS,JMSP,IIN,IOUT,IOTT
      DIMENSION X(20)
      DIMENSION A(6,6)
      LOGICAL FLAG
      DATA (A(1,I),I=1,6)/3*1. , 3*0./
      DATA (A(2,I),I=1,6)/3*1. , 3*0./
      DATA (A(3,I),I=1,6)/3*1. , 3*0./
      DATA (A(4,I),I=1,6)/3*0. , 3*1./
      DATA (A(5,I),I=1,6)/3*0. , 3*1./
      DATA (A(6,I),I=1,6)/3*0. , 3*1./
      IF (FLAG) GO TO 500
      PRINT(IOUT,*)" SEPERABLE TEST FUNCTION"
      FLAG = .TRUE.
500   CONTINUE
      F = 0.
      DO 55 I = 1,6
      DO 55 J = 1,6
      F = F + A(I,J) * X(I) * X(J)
55    CONTINUE
      OBJT3 = F
      RETURN
      END
```

Problem: U-17

```
      REAL FUNCTION OBJT4(X,K)
      COMMON/PRINT/IBRKT,IPOW,IMS,JMSP,IIN,IOUT,IOTT
      DIMENSION X(20)
      DIMENSION A(6,6)
      LOGICAL FLAG
      DATA (A(1,I),I=1,6)/3*1. , 3*.01/
      DATA (A(2,I),I=1,6)/3*1. , 3*.01/
      DATA (A(3,I),I=1,6)/3*1. , 3*.01/
      DATA (A(4,I),I=1,6)/3*.01 , 3*1./
      DATA (A(5,I),I=1,6)/3*.01 , 3*1./
      DATA (A(6,I),I=1,6)/3*.01 , 3*1./
      IF (FLAG) GO TO 500
      PRINT(IOUT,*)" SEPERABLE TEST FUNCTION"
      FLAG = .TRUE.
500   CONTINUE
      F = 0.
      DO 55 I = 1,6
      DO 55 J = 1,6
      F = F + A(I,J) * X(I) * X(J)
55    CONTINUE
      OBJT4 = F
      RETURN
      END
```

Problem: U-18

```
      REAL FUNCTION OBJT5(X,K)
      COMMON/PRINT/IBRKT,IPOW,IMS,JMSP,IIN,IOUT,IOTT
      DIMENSION X(20)
      DIMENSION A(6,6)
      LOGICAL FLAG
      DATA (A(1,I),I=1,6)/3*1. , 3*.1/
      DATA (A(2,I),I=1,6)/3*1. , 3*.1/
      DATA (A(3,I),I=1,6)/3*1. , 3*.1/
      DATA (A(4,I),I=1,6)/3*.1 , 3*1./
      DATA (A(5,I),I=1,6)/3*.1 , 3*1./
      DATA (A(6,I),I=1,6)/3*.1 , 3*1./
      IF (FLAG) GO TO 500
      PRINT(IOUT,*)" SEPERABLE TEST FUNCTION"
      FLAG = .TRUE.
500   CONTINUE
      F = 0.
      DO 55 I = 1,6
      DO 55 J = 1,6
      F = F + A(I,J) * X(I) * X(J)
55    CONTINUE
```

```
OBJT5 = F
RETURN
END
```

VITA

Daniel B. Fox was born on December 10, 1946 in Hinsdale, Illinois. He received his B.S. degree in Engineering Physics from the University of Illinois in January 1969. As a second lieutenant in the U.S. Air Force he completed a M.S. degree in Industrial Engineering - Operations Research at Oklahoma State University in the summer of 1970. While at O.S.U. he became a member of Alpha Pi Mu, the industrial engineering honorary fraternity. From 1970 to the spring of 1974 he worked as an operations research analyst on the director's staff of the National Security Agency. Captain Fox then attended Squadron Officers School at Maxwell Air Force Base, Alabama. In July 1974 he began work as a test engineer on a LORAN and inertial navigation and weapon delivery system for the Air Force at Eglin Air Force Base, Florida. In the summer of 1977 he entered the Ph.D. program in Operations Research in the Department of Mechanical and Industrial Engineering at the University of Illinois, Champaign-Urbana. He is currently employed as an assistant professor in the Operational Sciences Department of the School of Engineering at the Air Force Institute of Technology, Wright-Patterson Air Force Base, Ohio.

LMED

-8