In presenting the dissertation as a partial fulfillment of the requirements for an advanced degree from the Georgia Institute of Technology, I agree that the Library of the Institute shall make it available for inspection and circulation in accordance with its regulations governing materials of this type. I agree that permission to copy from, or to publish from, this dissertation may be granted by the professor under whose direction it was written, or, in his absence, by the Dean of the Graduate Division when such copying or publication is solely for scholarly purposes and does not involve potential financial gain. It is understood that any copying from, or publication of, this dissertation which involves potential financial gain will not be allowed without written permission.

7/25/68

CONSTRAINED NONLINEAR OPTIMIZATION

A THESIS

Presented to

The Faculty of the Division of Graduate

Studies and Research

by

Mark Henry Machina

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in the School of

Industrial and Systems Engineering

Georgia Institute of Technology

June, 1971

CONSTRAINED NONLINEAR OPTIMIZATION

Approved:

Chairman

Date approved by Chairman: June 10th 1971

ACKNOWLEDGMENTS

The author wishes to express his deep appreciation to Dr. C. M. Shetty for his infinite patience and valuable guidance in the preparation of this thesis. The author also thanks Dr. J. J. Jarvis, Dr. D. E. Fyfe, and Dr. D. C. Montgomery for their encouragement and assistance. Special thanks are extended to Professor J. J. Goda, Jr. for his invaluable assistance in preparing the computer program.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF ILLUSTRATIONS

SUMMARY

The Quadratic Programming algorithm of Theil and Van de Panne and its extension by Geoffrion for reducing a nonlinear inequality constrained problem to a sequence of simpler equality constrained subproblems are investigated to determine the feasibility of solving problems with nonlinear constraints in a combinatorial manner. This is found to be computationally successful, although no theoretical proofs are given.
It is also shown that, by relaxing the exactness with which each subproblem is solved, the algorithm still is successful and the efficiency of the computer program is greatly enhanced from the standpoint of execution time. It is also shown that it is advantageous to use the approximated solution to one-subproblem as the starting point for certain succeeding subproblems. The solution procedure is illustrated by an example problem and a computer program is given.

CHAPTER I

INTRODUCTION

Constrained nonlinear optimization refers to the determination of the optimal solution to the problem

$$\text{Maximize:} \quad f(x) \tag{1.1}$$

$$\text{Subject to:} \quad g_i(x) \geqq 0, \qquad i = 1,\ldots,m \tag{1.2}$$

$$x \in X$$

where $f(x)$ and $g_i(x)$ are real valued functions defined on E^n and X is an arbitrary set in E^n. If $\bar{x}$ maximizes 1.1 subject to 1.2, then we will call $\bar{x}$ the <u>optimal solution</u> to the problem. All points satisfying expression 1.2 will be called <u>feasible points</u>.

The above problem reduces to a linear programming problem when f and g are linear and $X = \{x : x \geqq 0\}$. Effective solution procedures such as the simplex method are available for solving such problems. A natural extension of the above linear problem is the Quadratic Programming Problem where the function f is quadratic. Different approaches have been adopted to solve such a problem, e.g.

1. Adjacent extreme point methods which move from one extreme point of the constraint set to another. See, for example, Wolfe (41), Dantzig (7), and Van de Panne and Whinston (40). This approach is perhaps the most effective procedure for quadratic programming.

2. Optimizing along directions which lead to improved feasible points, e.g. Beale (1), Zoutendijk (45).

3. Solving a sequence of equality constrained problems, e.g. Theil and Van de Panne (39). An outline of this approach is given below since this study deals with the adaption of this method to a more general problem. A thorough discussion is given in Chapter II.

The Theil and Van de Panne method maximizes a strictly concave quadratic function subject to a convex set of linear inequality constraints. It is an iterative method in which the inequality constrained problem is solved using a finite sequence of equality constrained subproblems. The unconstrained problem is first maximized and, if this solution falls outside the feasible space, we identify those constraints it violates. Subsets of these violated constraints are then considered in a combinatorial manner and the function again maximized with each subset of constraints in equational form. Constraints that are violated by each new subproblem solution are then added to those already imposed. The subsets are increased in size in an iterative process until either the optimal is found or it is shown that no feasible solution exists. Theil and Van de Panne showed quadratic convergence for this method, that is, the solution procedure will find the optimal in a finite number of steps for the quadratic objective function.

Geoffrion (19) has extended the Theil and Van de Panne algorithm to a general concave nonlinear objective function and has suggested that the requirement for concavity might also be relaxed. However, the procedure still requires that the constraints be linear.

This thesis is directed toward the following three objectives:

a. The application of the combinatorial approach to second and higher order functions with constraints which may not be linear.

b. The Theil and Van de Panne procedure requires determination of the additional violated constraints at each stage and not the exact solution. Means of taking advantage of this property will be investigated.

c. Since each subproblem is very "similar" to the preceding one, it seems reasonable to use the optimal solution of one problem in solving the subsequent problem. We will investigate means by which this can be computationally done.

Since we are dealing with an inequality constrained problem, we will first look at means of solving such problems. In Chapter II we will discuss the combinatorial approach and its extension. Since the combinatorial approach solves the inequality constrained problem by the use of a sequence of equality constrained subproblems, a discussion of solution techniques for the equality constrained problem and a statement of the particular solution procedure adopted for this research are given in Chapter III. The flow charts for the solution procedure used and a discussion of the computer program appear in Chapter IV. Chapter V includes the computational findings and the conclusions and recommendations are given in Chapter VI. The problems solved and the computer program are given in the Appendices.

Literature Survey

It may be recalled that the nonlinear programming problem we are dealing with is an inequality constrained problem of the form:

Max $f(x) : x \in X,\ g_i(x) \geqq 0, \quad i = 1,\ldots,m$. In this section, we will discuss some of the important methods available, both numerical and analytical, for solving this problem. Since some of the numerical methods are based on converting the problem to an equivalent unconstrained problem, we will first discuss the methods available for solving an unconstrained problem.

Unconstrained Maximization

Unconstrained maximization is accomplished generally by an iterative search which uses the relation

$$x_{i+1} = x_i + h_i d_i \tag{1.3}$$

where d_i is an n dimensional direction vector and h_i is a distance moved along it so that

$$f(x_{i+1}) \geqq f(x_i) \tag{1.4}$$

The basic scheme can be summarized as follows: At some iteration we are given a direction d_i. From a point x_i we proceed along d_i to a point $x_{i+1} = x_i + h_i d_i$. At x_{i+1} we determine a new direction d_{i+1} and repeat the procedure.

Iterative optimization techniques can be classified generally into two categories: gradient free methods and gradient methods. Gradient free search methods are those methods not requiring explicit evaluation of any partial derivatives of the function, but rely solely on values of the objective function f along with information gained from earlier iterations.

Some of the algorithms based on the above scheme are discussed below.

Cyclic Coordinate Method. In this method, the directions d_i are the coordinate directions. These directions are the same for every n^{th} iteration (i.e., $d_i = d_{i+n}$). The step length h_i along direction d_i is found by optimizing f along d_i.

Sequential Simplex Method. In this method, the direction of search is determined at each stage and this direction changes at each iteration. However, the step length at each iteration is fixed. More specifically, this technique (1) creates a regular geometric figure, called a simplex, (2) experiments at the vertices of the figure, and (3) moves away from the worst experimental point through the center of the figure locating a new experimental point at the mirror image of that point just rejected. As the search nears the optimal, the size of the simplex is reduced until it is adequately small to give an acceptable estimate of the optimal. The basic simplex method has been modified by Nelder and Meade (27) and Box (3) to include acceleration of the search when successes are encountered. These modifications will be considered later in this chapter when the inequality constrained problem is discussed.

Hooke and Jeeves Pattern Search. In this method, again the direction of search is determined at each stage based on local explorations. This direction changes from iteration to iteration, and the step length is varied to reward success in the direction of search. The details of the procedure are as follows:

Starting from some feasible base point, which we will call x_1, local explorations are made at some δ distance to either side of the base point in all n directions. If improvement of the functional value is experienced, the base point is moved to this new location, and its subscript advanced by 1. When the local exploration phase of the method is concluded, the newest temporary base point would be x_n. If at this time x_n is different from x_1, a step is taken in the direction $(x_n - x_1)$. The step length is some constant, c, times this distance, that is, the step length is $c(x_n - x_1)$. If the new base point established after this step shows improvement, the method is restarted from that point. If no improvement is found, the last temporary base point that showed improvement is taken as the new base point and the method restarted. If at the end of the exploration phase $x_n = x_1$, the distance δ is reduced and the method restarted. When δ is sufficiently small, we assume that we have found the optimal.

Powell's Conjugate Gradient Algorithm. Here again, the direction of search changes from iteration to iteration; however, the attempt is to obtain n mutually "conjugate" directions of search. The step length is determined by optimization along the direction of search. The conjugate directions are important since it can be shown that, if we optimize along n conjugate directions, we will reach the optimal when the objective function is quadratic. The basis of the method used to generate the conjugate directions is that, if we optimize a quadratic function along a direction α (starting from two different points) to give points x_1 and x_2, then α and $(x_1 - x_2)$ are mutually conjugate.

Rosenbrock Method. In this method the direction of search d_i is determined so as to align it along the axis of ridges or valleys based on the results of past success in local searches. The distance of movement also changes from iteration to iteration.* Some details of the procedure are as follows. For a problem with n variables, n orthonormal directions are used. Initially, unit vectors are used along the coordinate axes and, after initial exploration, a new set of directions is determined that is orthogonal to the previous set. The sequence of searches along each of these new directions is repeated. Whenever a success is followed by a failure, new directions are computed from the old and the aggregate results of each successful evaluation. Success is rewarded by increasing the step length in the successful direction by some factor greater than one and failure by multiplying the step length in a direction that fails by some negative factor less than one. Success is defined as an exploration resulting in a functional value that is greater than or equal to the previous value. One drawback of the Rosenbrock method is that, if too long a step is made, the search must back-up much more slowly with a series of shorter steps, each having n local searches. This is time consuming and detracts from the efficiency of the method. The modification of Davies, Swann, and Campey helps eliminate this deficiency by maximizing in each direction, thus avoiding the excessive step length.

---

*Davies, Swann, and Campey (38) have considered a modification using optimization along the direction of search. However, computational results show the modification gives no improvement in the convergence property.

Computational experience has shown that the above methods generally improve in the order in which they were presented, with the Cyclic Coordinate method being the least desirable and the Rosenbrock method being, perhaps, the most desirable. This is attributed to the fact that the Rosenbrock method permits change in step length and direction to accelerate convergence.

We now turn our attention to gradient methods. Gradient methods are generally accepted as being the more powerful, although other considerations sometimes make a gradient method undesirable. Setup time can often be a drawback since gradient methods are not as straightforward as the gradient free search procedures making them more difficult and time consuming to program. In addition to this, they are not as flexible as the gradient free search methods, as some functions are not differentiable or the gradient may not be available in closed form. In such a case, it is necessary to determine them by local exploration using several experiments, a procedure that in itself is time consuming. The effort spent along this line can outweigh the benefits of using the gradient search technique. These considerations and others discussed in Chapter III lead us to the use of a gradient free unconstrained search procedure in this study. Therefore, we will discuss below only the Davidon-Fletcher-Powell Method (18) which is considered to be the most powerful among the gradient algorithms.

In the gradient methods, the direction d_i in 1.3 depends on the partial derivatives of the objective function, f, with respect to the independent variables. The Davidon-Fletcher-Powell Method (18) is an improved version of Davidon's method (8). It is based on the idea of generating the inverse of the matrix of second partial derivatives of the function at the optimal point by a series of searches. This matrix, called the Hessian matrix, will be denoted by H. This is accomplished without the use of the second partial derivatives. An outline of the method is as follows.

At the i^{th} stage of the procedure we are given a feasible point x_i and an approximation H_i to the Hessian at the optimal point. The point x_{i+1} is found by optimizing f(x) in the direction p_i where

$$p_i = H_i q_i(x) \tag{1.5}$$

where $q_i(x)$ is the gradient of the objective function at x_i.
Letting

$$\beta_i = x_{i+1} - x_i \tag{1.6}$$

and

$$y_i = q_{i+1} - q_i \tag{1.7}$$

the approximation to the Hessian is changed to

$$H_{i+1} = H_i + A_i + B_i \tag{1.8}$$

where

$$A_i = - \frac{\beta_i \beta_i^T}{\beta_i^T y_i} \tag{1.9}$$

and

$$B_i = \frac{- H_i y_i y_i^T H_i}{y_i^T H_i y_i} \tag{1.10}$$

The procedure is started with some feasible point x_0 and initial approximation $H_0 = I$, an identity matrix. The procedure is stopped when the step length $||\beta_i||$ becomes sufficiently small, where $||\beta_i||$ is the norm of β_i.

Inequality Constrained Nonlinear Problems

In this section we will look at some of the methods of solving the nonlinear problem with only inequality constraints. The classical approach to this problem is via the Lagrangian function defined by

$$F(x,\lambda) = f(x) + \sum_{i=1}^{m} \lambda_i (g_i(x) - s_i) \tag{1.11}$$

where s_i is the slack variable associated with the i^{th} constraint and λ_i is the Lagrange Multiplier associated with the i^{th} constraint. Here the objective function, f(x), is penalized by any violated constraint, as $g_i(x) < 0$ when violated. Taking partial derivatives of the above defined function (1.11) and then solving the simultaneous equations resulting, we are able to determine the stationary point. However, solving the simultaneous equations is difficult for large problems. In recent years attention has been directed at methods such as the "generalized Lagrangian multiplier" approach (12).

For certain specially structured problems, we also have special methods of solution which have been found to be computationally very efficient; for example, the simplex method for linear programming and the simplex-like procedures for quadratic programming.

Yet another approach is to solve the nonlinear programming problem by solving a series of simpler problems. One such approach is the "combinatorial approach" which is the subject of this investigation and will be dealt with in detail in later chapters. Another approach is via "penalty functions" where we solve a series of unconstrained problems of the form

$$F = f(x) + \sum_i P(g_i(x)) \qquad 1.12$$

where the $\sum_i P(g_i(x))$ term is the penalty term that penalizes the function if the constraints are violated. This approach to solving the constrained nonlinear problem can be divided into two classes. Interior penalty function methods are those which start from a feasible point and approach the optimal at the boundary of the feasible space as if it were a barrier. Exterior methods are those which start from some point outside the feasible space, normally the solution to the unconstrained problem, and then proceed to close on the optimal from outside the feasible region. In the exterior methods, the objective function includes only those constraints that are violated.

There are several interior methods, some of which have been in use for several years. The most widely known and used of the interior methods is Fiacco and McCormick's (13) SUMT (Sequential Unconstrained Minimization Technique), which is a modification of the Created-Response Surface Technique of Carroll (6). Another interior method is due to Zangwill (44).

Since all of these approaches are similar, we will look at Fiacco and McCormick's SUMT as an example of interior penalty function methods as applied to a maximization problem.

SUMT is based on the transformation

$$F(x,r) = f(x) + r \sum_{i=1}^{m} 1/g_i(x) \qquad 1.13$$

where r is a sequence of decreasing values, $r > 0$. The method begins with the location of a feasible start point. F(x,r) is then minimized for succeeding decreasing values of r. As r approaches zero, the value of F(x,r) approaches that of f(x), since the penalty term decreases towards zero. Thus, at the optimal, the values of F(x,r) and f(x) are equivalent and both the penalty function and the objective function reach minimums simultaneously. One of the most serious shortcomings of this method is the difficulty encountered in the selection of the initial value of r and the rate at which it should be decreased, as the product of a very small number, r, and a very large number, $1/g_i(x)$, can cause difficulty in the convergence of the method.

Exterior methods are relatively new in the field of nonlinear optimization. In 1967, exterior techniques were introduced by Fiacco and McCormick (14) and Zangwill (43). In 1968, Lootsma (25) presented a combination of the interior point methods and the exterior methods for solving the constrained nonlinear problem and also in 1968, Powell (29) introduced another exterior method which appears to be the best attempt thus far.

Powell uses the transformation

$$F(x,r,s) = f(x) + \sum_{i=1}^{m} (g_i(x) + s_i)^2/r_i \qquad 1.14$$

where s and r are sequences of decreasing values with $r > 0$ and $s < 0$. In all of the penalty function methods mentioned, F is minimized for a sequence of values of r, giving a sequence of minimums that close on the true minimum. In those methods other than Powell's, F and f are equal at the optimal solution. Powell has added the second parameter s_i to reduce the difficulties encountered with the product of large and very small numbers near the optimal. Thus, in this method it is not necessary for F(x,r,s) to equal f(x) at the optimal solution, rather they must simply reach their respective minimums at the same time, i.e. if $\bar{x}$ minimizes F(x,r,s) then $\bar{x}$ minimizes f(x) also. Notice that both parameters r and s are subscripted to correspond with the constraints $g_i(x)$. This allows them to be reduced independently so that only those parameters corresponding to the constraints not converging to zero fast enough need be reduced. This allows those parameters whose constraints are converging sufficiently fast to remain unchanged, thus speeding the overall rate of convergence. When it becomes necessary to reduce r, it is accomplished by the following relation

$$r_i = r_i/10 \qquad 1.15$$

where the factor of 10 is arbitrary, but recommended by Powell. If the i^{th} constraint is converging fast enough, the parameter s_i is reduced as follows

$$s_i = s_i + g_i(x) \qquad 1.16$$

Recall that only those constraints which are violated are included in the penalty term and, therefore, the $g_i(x)$ is less than zero and s_i is monotonically decreasing. If the i^{th} constraint is not decreasing to zero fast enough, both r_i and s_i are decreased together, both by the factor of 10. A flow chart and further discussion of Powell's method can be found in Figure 4 and Chapter IV.

There are also several numerical methods that have been reasonably successful in solving nonlinear programming problems. These are extensions of gradient and gradient free methods discussed earlier. Some of the gradient free search methods discussed in an earlier section have been useful in solving the nonlinear constrained problem, for example, the Hooke and Jeeves Pattern Search (23). In this technique, fixed search directions and step lengths are used. When applied to the constrained problem, each test point is checked for feasibility. Should such a point prove infeasible, a different search direction is tried. If all search directions giving improvement lead to infeasible points, the step length is shortened and the same directions tried. Due to the fixed directions of search, this technique may fail to find the true optimal since the search will be halted when the step length becomes sufficiently small and, if we reach a point where the only directions giving functional improvements lead to infeasible points, the search will be stopped even though the optimal has not been found.

The Sequential Simplex of Spendley, Hext, and Himsworth (35), also discussed earlier is another method that has been extended for constrained optimization. This method is different from the pattern search technique in that the direction of search is not fixed. With inequality constraints, each new vertex must be checked for feasibility. When an infeasible one is encountered, it is assigned a large negative value which penalizes it enough to cause the search to reflect back in a feasible direction. Should all possible directions offer infeasible vertices, the length of the sides of the simplex is decreased and the search continued.

Nelder and Mead (27) have modified the above method to include an expansion and contraction of the simplex to award success by extending the simplex in the successful direction and punish failure by contracting the simplex in directions which fail to bring improvement in the functional value. Should the contraction fail to bring improvement, the size of the entire simplex is reduced.

The sequential simplex method has also been modified by Box (3) who named his new modification the Complex method. It differs from the simplex method in that there are $k > n+1$ points in the figure that is created. The sides of the figure are not necessarily of equal length. Once again, the vertex with the worst reading is rejected and reflected through the centroid of the figure, but some $\alpha > 1$ times as far from the centroid as the rejected point, to establish a new point. Should this point be infeasible, it is moved back, halfway towards the centroid. This process is repeated as many times as necessary until a feasible point is found. Thus, as we would expect, the complex method tends to flatten out along the binding constraint. The complex can then move along the constraints to the optimal. It stops when five consecutive evaluations give the same functional value within the acceptable tolerance, which means the complex has essentially collapsed into its centroid. An important advantage of the complex method over the simplex is exactly the relaxation of the requirement for a regular geometric figure. Starting procedures are also easier due to this property since only one feasible point need be found and the irregular figure is constructed from this one point.

Powell's conjugate direction method is not suitable for use with constrained problems since the solution to such problems is likely to lie on a boundary and the basis for the effectiveness of conjugate direction methods is the existence of an optimal at a stationary point. It is in that situation that the function can be approximated by the quadratic form.

Rosenbrock's unconstrained search, on the other hand, can be successfully applied to constrained problems. The procedure starts with a feasible point and proceeds in the same manner as the unconstrained search technique, except that each new point is tested for feasibility. A "boundary region" is defined along the boundary of the feasible space. When we detect that the search has entered or passed through the "boundary region," it is assumed that the function optimal probably lies outside the feasible region and the function is modified so that it will remain within the feasible region. The search is retracted a distance (depending upon the amount of penetration into the "boundary region") back towards the last feasible point encountered. The search is then continued and further modification to the function is made as the "boundary region" of other constraints is entered.

We will now consider some of the gradient methods of approaching the constrained nonlinear problem. Several such methods have been developed; however, no one best method exists and each seems to be better suited for a particular type problem. Those to be discussed here are the method of Glass and Cooper (20), Zoutendijk's method of feasible directions (45), Rosen's projected gradient method (31), and Davidon's method with linear constraints (18) as modified by Fletcher and Powell.

The method of Glass and Cooper is essentially a steepest ascent method that follows the gradient as far as possible. Starting from a feasible start point, we move in the direction of the gradient a predetermined distance s. If the functional value is improved and no constraints are violated, we continue in the same direction a distance cs where c is some constant greater than 1. This procedure is repeated until failure is encountered. If the failure is due to a poorer functional value, the last successful point is used as a new base point and a new direction determined. If the failure is due to a constraint violation, a new base point is established some δ distance inside the binding constraint and a new rule for the selection of search direction is adopted, since the gradient takes us outside the feasible space. The step length s is reduced and shorter moves are taken along the binding constraint. When the point is found from which no direction offers improvement in the functional value, we have arrived at a local optimal.

Zoutendijk's method of feasible directions is restricted to problems with linear constraints only. It also starts from a feasible point and proceeds in a direction determined by linearizing the objective function in the vicinity of the start point and solving the linear programming problem. This direction is the feasible direction which makes the smallest possible angle with the gradient at that point and offers the greatest possible improvement in the objective function. Once the search direction has been determined, a one-dimensional search is conducted to determine the optimal in that direction. A large step is then taken to the optimal in that direction if one exists, or to the first binding constraint encountered. In either case, a new base point is thus located and the procedure repeated. When there exists no direction in which functional improvement can be gained, we have located a local optimal.

The gradient projection method of Rosen is different from the preceding two methods in that rather than search around the interior of the feasible space, it moves along the boundaries from the start. If equality constraints are present in the problem, this method starts from their intersection and proceeds as directed by the projection of the gradient of the objective function. If equality constraints are not present in the problem, a feasible start point is chosen and the gradient followed directly until one or more constraints are binding. The projection of the objective function gradient is then taken on the intersection of binding constraints. This direction is followed until the next binding constraint is found. At that time the procedure is repeated and we continue in this manner until the optimal is located.

The Davidon-Fletcher-Powell method has also been applied to constrained problems. Recall from the previous discussion of this method that the i^{th} direction of search is obtained from the product of the i^{th} approximation of the Hessian and the i^{th} gradient of the function, i.e. $p_i = H_i q_i(x)$. The basic difference in the method when applied to constrained problems is in the calculation of this direction, p_i. The constraints are taken into consideration in the formulation of the approximation of the Hessian, so that if k constraints are binding at a particular stage, the new direction is determined by $p_i = H_{i_k} q_i(x)$ where H_{i_k} is the new approximation of the Hessian which will yield a feasible direction taking the constraints, k, into account.

We will now proceed with a discussion of the combinatorial approach of Theil and Van de Panne for solving the constrained quadratic problem and Geoffrion's extension of it to include problems of higher order than the quadratic.

CHAPTER II

THE COMBINATORIAL ALGORITHM

Theil and Van de Panne's Quadratic Programming Algorithm

Perhaps the first combinatorial approach for solving nonlinear programming problems is that proposed by Theil and Van de Panne (39) for maximizing a strictly concave quadratic function subject to linear independent, inequality constraints. Dependent constraints can give rise to the degenerate case and, therefore, Theil and Van de Panne assume all constraints are independent. As discussed in Chapter I, it is an iterative procedure in which they consider a finite sequence of equality constrained subproblems beginning with the unconstrained problem and continuing with additional subproblems, each considering, in equational form, a subset of constraints. The sequence of subproblems continues until either the optimal is found or it is shown that no feasible solution exists. The combinatorial approach of Theil and Van de Panne and Geoffrion's extension of it will be discussed in detail below, since this study is concerned with testing its computational feasibility for more general problems.

It will be helpful to begin with the definition of some notation

M : the set of all constraints = $\{1,2,\ldots,m\}$

S : the set of constraints held in equational form in each subproblem, $S \subset M$, called a Trial Set

P_S : the subproblem corresponding to a set $S \subset M$:

Maximize: $f(x)$

Subject to: $g_i(x) = 0,\ i \in S$

x^S : the solution to P_S

T_S : those constraints in $(M - S)$ that are violated by x^S

$T_S = \{i \in M\text{-}S:\ g_i(x^S) < 0\}$

$\bar{S}$: the set of constraints satisfied as equalities at the optimal solution, $\bar{x}$, to the nonlinear programming problem defined by equations 1.1 and 1.2.

We will now discuss the method proposed by Theil and Van de Panne (39) to solve a quadratic programming problem. The method is based on the following three rules.

Rule 1: If x^o (the vector of the unconstrained optimal) violates certain constraints, then $\bar{x}$ (the optimal vector) satisfies at least one of these exactly.

Rule 2: Suppose that two or more constraints are satisfied exactly by $\bar{x}$ and partition the set of these constraints into two subsets, S and S', containing at least one constraint each. Then x^S (the vector which "maximizes" F subject to the constraints in S in equational form) violates at least one constraint which is an element of S'.

Rule 3: Suppose that for some subset S of the constraints, x^S exists and violates none of the constraints; then $x^S = \bar{x}$ if and only if every x^{S^h} violates the h^{th} constraint, where

$$S^h = S - \{h\}$$

If $\bar{S}$ is known, then $x^{\bar{S}} = \bar{x}$, the optimal solution. Our attempt is to obtain $\bar{S}$ by solving a series of equality constrained problems. Suppose, at the k^{th} stage, we have a set U_k whose elements are k-element subsets of M. The elements of U_k are called the current generation of trial sets and we would like to test whether any element, S, of the set is equal to $\bar{S}$. Each such S is called a trial set. Recall that T_S denotes the constraints violated by x^S.

At some stage, if each element of U_{k-1} has been tested, we will be defining a new generation of trial sets. This is given by

$$U_k = \{\{S,t\} : S \in U_{k-1}, \quad t \in T_S\} \qquad 2.1$$

It may be noted that each succeeding generation of trial sets has one more element than the previous one.

The procedure starts with $S = \emptyset$ so that

$$U_o = T_\emptyset = \{i \in M : g_i(x^o) < 0\}$$

Figure 1 gives the flow diagram for the combinatorial approach and the following clarification may be helpful.

BLOCK 1: The solution procedure begins with the determination of the optimal of the unconstrained problem (1.1) where $S = \emptyset$. The solution vector x^o is then used to identify U. Should $U^o = \emptyset$, we have the case where the unconstrained optimal is within the feasible space and $x^o = \bar{x}$. When $U \neq \emptyset$ we begin the iterative process with Block 2.

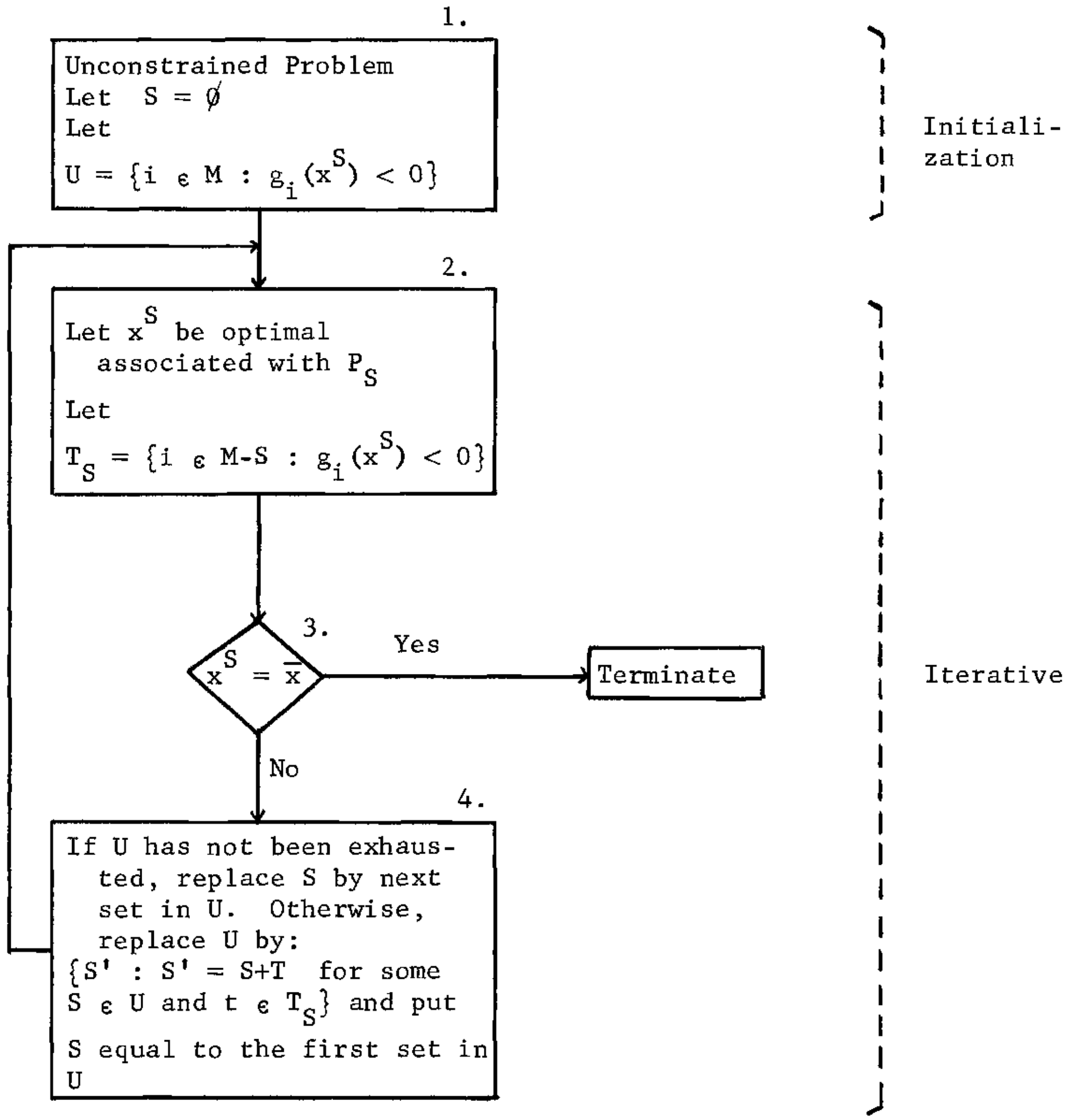

Figure 1. Flow Diagram for the Theil and Van de Panne Algorithm

BLOCK 2: Take a subset S of U and solve P_S for x^S. Now use x^S to identify T_S, those constraints (not in S) that are violated by x^S.

BLOCK 3: (Test x^S for optimality.) If $T_S = \emptyset$, we apply Rule 3, otherwise $x^S \neq \overline{x}$ and we move on to Block 4. If $x^S = \overline{x}$ we have solved the problem and terminate.

BLOCK 4: We choose another untested element S in U and return to Block 2. On the other hand, if all elements of U have been tested, we redefine U with a new generation of trial sets. Each element S_{k-1} of the previous generation of trial sets gives rise to one or more elements of the new generation of trial sets. The new elements are given by

$$S_k = S_{k-1} + t \ , \quad \text{where } t \in T_{S_{k-1}} \qquad 2.2$$

Now return to Block 2.

To illustrate the algorithm we will consider the example given in Figure 2.

The first step (Block 1) is to determine the unconstrained solution, x^o, and in Fig. 2 we see that x^o violates constraints 3 and 4. Therefore, U_o contains the subsets {3} and {4} which will now be considered as we move to Block 2.

In Block 2, we take the first subset of U, say {3} and solve our first subproblem with constraint 3 in equational form. The solution vector to this subproblem will be written $x^{(3)}$. We now use the solution vector, $x^{(3)}$, to determine which, if any, constraints it violates. Fig. 2 shows that it violates constraint 4.

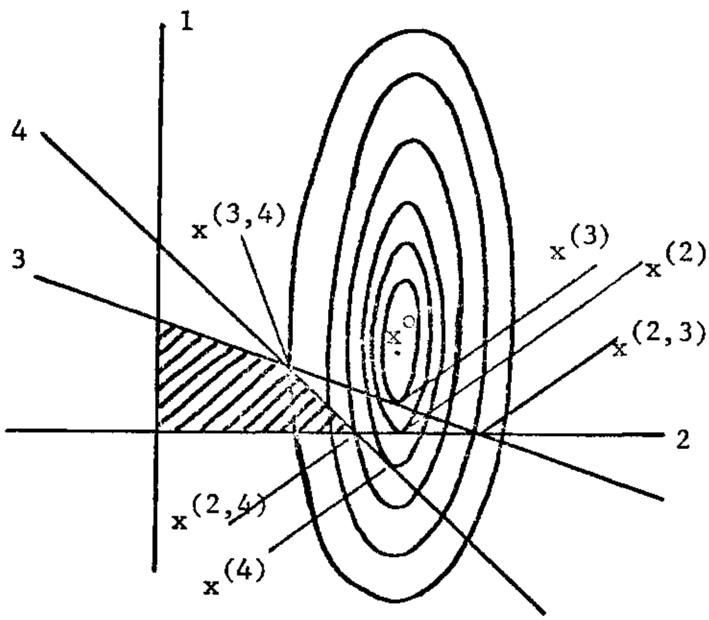

Figure 2. Sample Problem

Having a violation, we move through Block 3 to Block 4. Here we see that U is not exhausted and therefore return to Block 2 to consider constraint 4 in equational form.

Once again we have a violation, as $x^{(4)}$ violates constraint 2. Having not found the optimal, we move on to Block 4 and see that U has now been exhausted and must be redefined. The new generation of trial sets, U, now contains the elements $\{3,4\}$ and $\{2,4\}$.

In the next iteration we then solve for $x^{(3,4)}$ and $x^{(2,4)}$ and find that both may be optimal as neither solution vector violates any further constraints. Rule 3 is now applied and we first consider $x^{(3,4)}$ and observe that $\{3,4\}$ is the set of constraints satisfied in equational form, so the sets S^h to be analyzed are the set $\{3\}$, obtained by excluding constraint h = 4, and the set $\{4\}$, obtained by excluding h = 3. Hence we must verify whether it is true that $x^{(3)}$ violates constraint 4 and that $x^{(4)}$ violates constraint 3. An inspection of Figure 2 shows that this is the case for $x^{(3)}$, but not for $x^{(4)}$; this vector violates constraint 2, not 3, and we therefore have a feasible solution, but not the optimal. We move on to $x^{(2,4)}$, which satisfies constraints 2 and 4 exactly. Does $x^{(4)}$ violate constraint 2, and $x^{(2)}$ violate constraint 4? The answer is affirmative as seen in Figure 2 and we can therefore conclude that $x^{(2,4)} = \bar{x}$. While this result is obvious for so few variables, an algebraic device such as Rule 3 is necessary when we deal with more than a few variables.

Geoffrion's Extension of the Combinatorial Approach

As mentioned above, the method discussed was developed for quadratic objective functions. Geoffrion (19) presented an extension to the combinatorial approach by considering nonquadratic concave functions (1.1) with a set of linear inequality constraints. This extension also entails the solution of a sequence of equality constrained subproblems which terminates with the optimal solution $\bar{x}$ or with the conclusion that no feasible solution exists.

It may be recalled that, corresponding to a subset $S \subset M$, we have defined a subproblem P_S as

P_S: Maximize: $f(x)$ 2.3

Subject to: $g_i(x) = 0, \quad i \in S$ 2.4

$x \in X$

The Theil and Van de Panne approach solves for the solution x^S and looks at the violation of the constraints in M - S. In Geoffrion's approach, the Lagrangian multipliers (dual variables) associated with the above solution are also considered. If they are of the wrong sign, the corresponding constraint is deleted from the succeeding generation of trial sets. This ability to reduce the elements of the trial sets permits Geoffrion to start with an arbitrary trial set, S^o.

While the extension considers the nonquadratic concave function, only linear constraints are included in the iterative combinatorial execution of the solution procedure. Nonlinear constraints must be included in the definition of the set X.

The procedure begins by considering 2.3 and 2.4 where $S = S^o$; S^o being the initial subset of M to be considered in equational form and may be the null set or some subset of the constraints known to be satisfied as equalities at the optimal, $\bar{x}$. If U, the current generation of trial sets, does not contain the optimal, we redefine U. The first generation U_o equals S^o alone (i.e. $U = \{\{i\} : i \in S^o\}$). The next generation is defined by $S = S^o \pm t$ for some $t \in T_S$. At any particular iteration, say the k^{th} where $k \geqq 1$, $S_k = S_{k-1} \pm t$ for some trial set S_{k-1} in the $(k-1)^{st}$ generation and $t \in T_{S_{k-1}}$. The set S_{k-1} is called the immediate lineal predecessor of S_k and either $S_k \subset S_{k-1}$ or $S_k \supset S_{k-1}$. Obviously, S^o is a lineal predecessor of all trial sets. The decision to add or subtract t depends on the sign of the Lagrangian multiplier from the solution of the dual subproblem. If it is negative, t is subtracted from S_{k-1} as we have found an S_{k-1} such that $\bar{S} \subset S_{k-1}$, where $\bar{S} = \{i \in M : g_i(\bar{x}) = 0\}$. Essentially, the expression $S \pm \{t\}$ denotes $S \cup \{t\}$ when $\{t\} \not\subset S$

and $S - \{t\}$ otherwise. The iterative process of defining U, then testing its elements for optimality and redefining U continues until we are able to find the optimal combination of equational constraints, $\overline{S}$, or we determine that no feasible solution exists. Normally, if S^o differs from $\overline{S}$ by more than a half dozen indices, the technique fails to be computationally efficient. If there are only a few constraints in the problem or it is known that only a small number are in $\overline{S}$, then $S^o = \emptyset$ can be a satisfactory starting subset.

Now suppose $\overline{x}$ is the optimal solution to the nonlinear programming problem with associated values λ_i of the optimal Lagrangian multipliers. For convenience in the discussion that follows, we will assume $\lambda_i > 0$ for $i \in \overline{S}$ where $\overline{S} = \{i \in M : g_i(\overline{x}) = 0\}$. Clearly, we have $x^{\overline{S}} = \overline{x}$. We will denote by $\mu(K)$ the number of elements in a set K, e.g. for K = 1,3,5 $\mu(K) = 3$.

We would like to define a "distance" between S^o and $\overline{S}$ which is correlated to the computational efficiency of the combinatorial approach. Such a measure is given by the following definition of distance d.

$$d(S^o,\overline{S}) = \mu(S^o - \overline{S}) + \mu(\overline{S} - S^o) \qquad 2.5$$

Geoffrion (19) has shown that, starting from S^o, the optimal subset, $\overline{S}$, of constraints is obtained in exactly $d(S^o,\overline{S})$ generations of trials. From experimental results, we know that, as $d(S^o,\overline{S})$ increases, the number of subproblems required to reach the optimal increase very rapidly. From this it is clear that the combinatorial approach is not practical if the optimal subset of equational constraints is very different from S^o. When considering the Theil and Van de Panne algorithm where we always have $S^o = \emptyset$ and constraints are added one at a time, this can be interpreted as saying that, as the optimal subset of equational constraints becomes large, the efficiency of the method decreases rapidly.

While the extension to the combinatorial approach is primarily concerned with the strictly concave f(x), as suggested by Geoffrion (19), it may be possible to apply this technique to the nonconcave f(x) as well. Possible modification of the algorithm to address the nonconcave function might be the setting of T_S equal to the indices of the constraints that are violated by any sequence $<x^v>$ feasible in P_S for which $<f(x^v)> \to \infty$. That is, violated by a sequence of points for which the functional value, $f(x^v)$, is unbounded, but which are feasible for the particular subproblem, P_S, at hand. Further discussion of the problem of nonconcavity and/or nonconvexity appears later.

The solution procedure used in this research uses a numerical algorithm for the solution of P_S which does not yield the Lagrangian multiplier used by Geoffrion to redefine his generations of trial sets. Without the knowledge of the Lagrangian multiplier, it was necessary to follow the Theil and Van de Panne algorithm of starting with $S^o = \emptyset$ and redefine U via

$$U = \{S' : S' = S + t, \quad \text{for some } S \in U \text{ and } \{t\} \in T_S\}$$

as shown in Block 4 of Figure 1.

CHAPTER III

SOLVING EQUALITY CONSTRAINED PROBLEMS

As seen from Chapter II, the Theil and Van de Panne procedure requires us to solve a sequence of equality constrained problems. If we begin with the solution of the unconstrained problem with $S = \emptyset$, we normally find ourselves outside the feasible space. Solving the sequence of combinatorial subproblems then brings us back to the point on the feasible space boundary that is the optimal point. In this chapter we will discuss the means used to solve these constrained subproblems.

One of our objectives was to take advantage of the fact that each subproblem differs from its lineal predecessor by only one constraint. Because of this "closeness" between the problems, it seems reasonable that the solution to one subproblem would be a good start point for its successor. This is facilitated by adopting a numerical solution procedure rather than an analytical method (even if one were available) for solving the subproblem, P_S. Additionally, numerical methods are more easily programmed than analytical methods.

Consider, again, the subproblem $P_{S_{k-1}}$ in the $(k-1)^{st}$ iteration of some subproblem where the trial set S_{k-1} of constraints are held to equalities. Recall from Chapter II that, when moving to the k^{th} iteration, S_k was constructed by the addition of one constraint to S_{k-1} by: $S_k = S_{k-1} + t$, where $t \in T_{S_{k-1}}$. Now if the solution $x^{S_{k-1}}$ to the subproblem $P_{S_{k-1}}$ is used as the start point for P_S, we see that this start point is "exterior" to P_{S_k} since S_k contains the elements of S_{k-1} plus an additional constraint that was violated by $x^{S_{k-1}}$. By "exterior" we mean outside the feasible space of P_{S_k}. Thus, each subproblem is solved starting from a point that is exterior to its feasible space. It is this precise point that governs our selection of numerical solution techniques. Those techniques requiring a feasible start point were eliminated from consideration in view of this. However, certain penalty function methods do start from an infeasible point and are discussed below.

Penalty function methods essentially solve a sequence of unconstrained problems whose values tend toward the true value of the objective function. The unconstrained problem has the form

$$F = f \pm \sum_i P(g_i) \qquad 3.1$$

where the term $\sum_i P(g_i)$ is a penalty term that is a function of the constraints and that drives the value of the penalty function F towards the true constrained optimal. Once the penalty function F has been defined, one of the unconstrained optimization techniques can be used to solve it.

Fiacco and McCormick's technique uses the transformation

$$F(x,t,r) = f(x) - r^{-1} \sum_{i=1}^{m} (g_i(x) - t_i)^2 \qquad 3.2$$

where f(x) is the original function to be optimized, $g_i(x)$ represents the constraints, r is a monotonic decreasing sequence approaching zero, and t_i is the i^{th} non-negative slack variable. A sequence of subproblems is then solved, each with decreasing values of r. The solutions to this sequence move closer to the true optimal as r is decreased.

Zangwill's method is a variation of the above and uses the form

$$F(x,r) = f(x) - r^{-1} \sum_{i=1}^{m} \text{Min}(g_i(x), 0)^2 \qquad 3.3$$

Here again, r is a monotonic decreasing sequence approaching zero and $f(x)$ and $g_i(x)$ have the same significance as in (3.2).

Both of the above methods can be used with equality as well as inequality constraints and are based on the idea that, as the parameter r decreases toward zero, the penalty term also reduces to zero. Thus, the entire penalty function approaches the value of the original function being optimized as we close in on the true optimal. It is here that the difficulty arises and the selection of r is critical as the product of a very large number, $1/r$, and a very small number, g_i, tends toward zero. Minimization under these circumstances is often difficult.

To overcome this problem, Powell (29) suggested that it is necessary for the penalty function, F, and the original function, f(x), to have their minima occur at the same point but that they need not be equal at that point. To accomplish this, a second parameter, s, is added to the penalty term, thereby reducing the sensitivity in the selection of r which is present in Fiacco and McCormick's and Zangwill's methods. The transformation used is

$$F(x,r,s) = f(x) + \sum_{i \in T_S} (g_i(x) + s_i)^2/r_i \qquad 3.4$$

where T_S is as defined in Chapter II. Again, r is a sequence of decreasing values tending toward zero. The parameter s is a decreasing negative value. Notice that both parameters r and s are subscripted so that each constraint has associated with it a parameter r and s. Since only those constraints that are violated are included in the penalty term, this allows selective reduction of the parameters to assist convergence of the particular subproblem being considered without affecting the parameters associated with constraints not included in the current subproblem being solved. It further allows the reduction of only those parameters associated with constraints that are not converging to zero at a satisfactory rate as the penalty function tends toward the true optimal. As in the previously mentioned penalty function methods, the penalty term includes the square of the constraints involved to insure continuity and differentiability. This also increases the probability of finding a global minimum. A flow chart of the Powell penalty function method appears in Figure 4 found in Chapter IV along with a more detailed discussion of the method. At this point, it is sufficient to say that this property of a set of parameters for each constraint makes the Powell method desirable to use in conjunction with the combinatorial approach. Additionally, Sasson (34) reports successful application of the Powell algorithm and states that it is more desirable than those of Fiacco and McCormick or Zangwill. For these reasons, it was decided to apply the Powell penalty function method in the solving of the subproblems of the combinatorial approach.

With this choice of penalty function method, we have now to choose an unconstrained optimization technique to optimize the penalty function, F. Box, Davies, and Swann (5) report that, when using gradient methods for optimizing the penalty functions, one can encounter serious problems since the penalty functions introduce steep valleys or ridges. Discontinuities may also arise in the second derivatives of the penalty function. Therefore, a gradient free method was desirable for the solution of the unconstrained problem produced by Powell's penalty function.

In Chapter I, several gradient free techniques were discussed that could be used to solve the unconstrained problem. One of the methods discussed was that due to Rosenbrock (33) along with its modification due to Davies, Swann, and Campey (38). This technique has been compared by Fletcher (17) with other unconstrained methods and is considered to be favorable over Powell's conjugate direction method when the number of variables is large and generally better compared with other approaches for solving unconstrained problems. In this study we have used Rosenbrock's unconstrained search for solving Powell's penalty function.

The procedure adopted in this study may, therefore, be summarized as follows. A sequence of equality constrained problems is formulated via Theil and Van de Panne's approach. These are converted into equivalent unconstrained problems using Powell's penalty function which, in turn, are solved using Rosenbrock's unconstrained search. A flow chart of the complete solution procedure appears in Chapter IV (Figure 3). Explanations of the block titles are given in the discussion of the program in Chapter IV.

CHAPTER IV

THE COMPUTATIONAL SCHEME

In the previous chapters we discussed briefly the techniques used in the solution of the constrained nonlinear problem (1.1) and (1.2). We shall now show how these techniques were fitted together to form the exact solution procedure used. We will discuss the decision rules used to take advantage of Theil and Van de Panne's approach of not solving each subproblem exactly. We will also present the test problems used.

To take advantage of Theil and Van de Panne's approach of not solving each subproblem exactly but only close enough to determine which, if any, constraints in the set (M - S) that particular subproblem violated, five different decision rules discussed below were tested. Each used different criteria for stopping the search in the subproblem. Rules 3 and 4 were tested at two levels of tolerance to see the effect of relaxing the exactness of the solution in each subproblem. Rules 1, 2, and 5 were run with four different levels of exactness. The attempt being made to relax exactness far enough to gain efficiency without identifying the wrong constraint in (M - S) as being violated. These are hueristic rules which we feel are useful in measuring the progress in convergence of each subproblem and can be stated as follows.
Discontinue the search when:

1. $\max_{i \in T_S} \{|g_i(x)|\} \leqq \delta$: This rule continues the search for a

more exact solution until the greatest constraint violation is less than some acceptable value, δ. The idea here is that, if the constraint in S with the greatest violation has been driven to within some small distance, δ, of zero, all the other constraints of S must be even closer to equalities and therefore the desired level of exactness has been reached in the solution.

2. $\sum_{i \in T_S} \{|g_i(x)|\} \leq \delta$: Here, rather than consider the greatest violation, the sum of all violations is considered. This rule prevents one constraint, which may be converging to zero slowly, from holding back the solution procedure when the other constraints of S may be at the exact solution. The sum of all constraint violations is driven to within some δ of the exact solution.

3. Max $|x_j^i - x_{j-1}^i| \leq \delta$ where $x_j = x_j^1 \ldots x_j^n$ is the solution at the j^{th} step: The step length taken in each of the n directions is measured here. When the largest step is less than δ, we know that the step lengths in the other (n - 1) directions is even smaller and the search is halted.

4. $\sum_{i \in n} |x_j^i - x_{j-1}^i| \leq \delta$: As in the second test rule, it is hoped that, if the step lengths in all but perhaps one or two directions are close to zero, we are close enough to the exact solution to determine T_S accurately. Therefore, the sum of the step lengths in the N directions is driven to within δ of zero.

5. $|f_j - f_{j-1}| \leqq \delta$: The search is halted in this case when the functional improvement resulting from the most recent step is less than δ. While it is possible that flat plateaus can "fool" this decision rule, presumably δ can be made small enough to avoid this in most cases. It is assumed that, when a step brings sufficiently small functional improvement, we are close enough to the exact optimal to determine T_S accurately.

The computer program was modified for each of the five decision rules and the following data were collected for each test problem and for various levels of desired exactness.

1. Execution time required.

2. Number of steps made (corresponds to the number of times the search routine was called).

3. Accuracy of the final solution.

Test Problems

Four test problems taken from the literature were used in this study and are listed in Appendix A. Problems P-1 through P-3 have quadratic objective functions with linear constraints in problems P-1 and P-3, and nonlinear convex constraint set in problem P-2. Problem P-4 is a fourth order polynomial with a saddle-point optimum and with convex nonlinear constraints. Problem P-5 in Appendix A is a third order polynomial with a nonconvex constraint set. This was used essentially to demonstrate the problems that arise in using the Theil and Van de Panne procedure for the case with nonconvex constraints.

The results of the analysis of these problems are presented and discussed in Chapter V.

Program Discussion

The program consists of a MAIN program which drives nine subprograms. Essentially it selects the constraint sets, S, to be held as equalities for each subproblem, creates the penalty function, and solves the now unconstrained problem for the solution vector, x^S. This x^S is then tested for optimality. If it is optimal, the program terminates, otherwise the next subproblem is solved by repeating the same process.

To assist in the explanation of the program, listed in Appendix B, it will be helpful to first define some terms used in the program.

CUTOF : The exactness with which we solve each subproblem, i.e. the δ distance from the exact optimal to which we drive the solution of each subproblem.

R : The initial value of the parameter r in Powell's penalty function. Read in from data card.

RN(I) : Updated value of the parameter r in Powell's penalty function.

S : The initial value of the parameter s in Powell's penalty function. Read in from data card.

SN(I) : Updated value of the parameter s in Powell's penalty function.

N : Number of variables in the problem at hand. Read from data card.

M : Number of constraints in problem at hand. Read from data card.

ITRMAX : Number of iterations in each Rosenbrock search.

ISTGMX : Number of stages permitted in each Rosenbrock search.

X(I) : The vector of the unknown variable.

K : The number of the iteration. Corresponds to the number of constraints in the current generation of trial sets.

TOTV : Total number of constraint violations for a given x^S.

VIOLAT(I,J): A zero/one matrix indicating a violated constraint by a one and a constraint not violated by a zero. The columns correspond to the M constraints and the rows to the set of current trial sets.

MOLD(I,J) : An "address" matrix whose rows identify those sets of constraints to be held as equalities in the current generation of subproblems. The number of non-zero columns corresponds to the iteration number, K.

IROW : A counter which indicates the number of rows in the VIOLATE and MOLD matrices which corresponds to the number of elements in the current U.

ICOUNT : A counter indicating the number of times the Rosenbrock search has been called.

AX(I) : A dummy variable used to save the solution to the unconstrained problem to be used as a start point for the subproblems of the first iteration.

BX(I,J) : A dummy variable used to save the solution to the first iteration subproblems to be used as start points for subsequent subproblems.

W(I) : A zero/one coefficient used to select those constraints identified in the MOLD matrix as part of the penalty function.

Initialization Step

The initialization step consists of moving from some start point to the unconstrained optimal and determining which constraints are violated at that point. Once initial values of various variables are inserted into memory, we are prepared to solve the unconstrained problem using Rosenbrock's method. This is accomplished by calling subroutine ROSENB, which is a program of the unmodified Rosenbrock search (11). ROSENB begins with the start point and takes its exploratory steps, evaluating the problem function by calling on subroutine FOFX, which has been loaded with the function statement. This function subroutine evaluates the function itself, constructs the penalty function (4.1), and evaluates it. In the initialization step we are considering the unconstrained case and, therefore, the penalty

$$\mathrm{FOFX} = \mathrm{FOFX} + \sum_{I=1}^{m} W(I)\ [(CI(I) + SN(I))^2/RN(I)] \qquad 4.1$$

function has no penalty term (i.e. $W(I)\ [(CI(I) + SN(I))^2/RN(I)] = 0$). The result is the solution x^o to the unconstrained problem after 1000 iterations of ROSENB. If the problem function is nonconvex and the solution to the unconstrained problem is unbounded, the program senses this when the functional value exceeds 10^{20} at which time ISWIT is set equal to 1 indicating that the unconstrained problem is unbounded, and we are returned to the MAIN program. Here x^o at the point of cutoff of the search is divided by 1000 and saved to be used as a future start point. (The choice of 1000 is arbitrary.) The solution, x^o, is now substituted into the constraints to determine violations. This is accomplished by calling subroutine CI(I) a functional subroutine that evaluates the constraints. Any constraint evaluation that is negative indicates a violation and another entry is made in the first column of the MOLD matrix. If all constraint evaluations are $\geqq 0$, we have an unconstrained optimal that is feasible and the problem is solved. If this is not the case, the initialization step is completed and we move on to the first iterative step and Block 3 of Figure 3.

Iterative Step

Each iterative step begins with the updating of the iteration counter K. If this counter exceeds M, the number of constraints in the problem, we know that no feasible solution has been found to this point and either the program has failed to find the true solution, no feasible solution exists, or the problem is of such a form, e.g. nonconvex constraint set, that the solution technique cannot solve it. The program is therefore halted in this case. When $K \leqq M$, we continue by addressing the first subproblem of the K^{th} iteration.

The current generation of trial sets of constraints to be held as equalities is stored in the MOLD array, each row identifying the trial set for one subproblem. Assume that, in some problem with M = 6, x^o

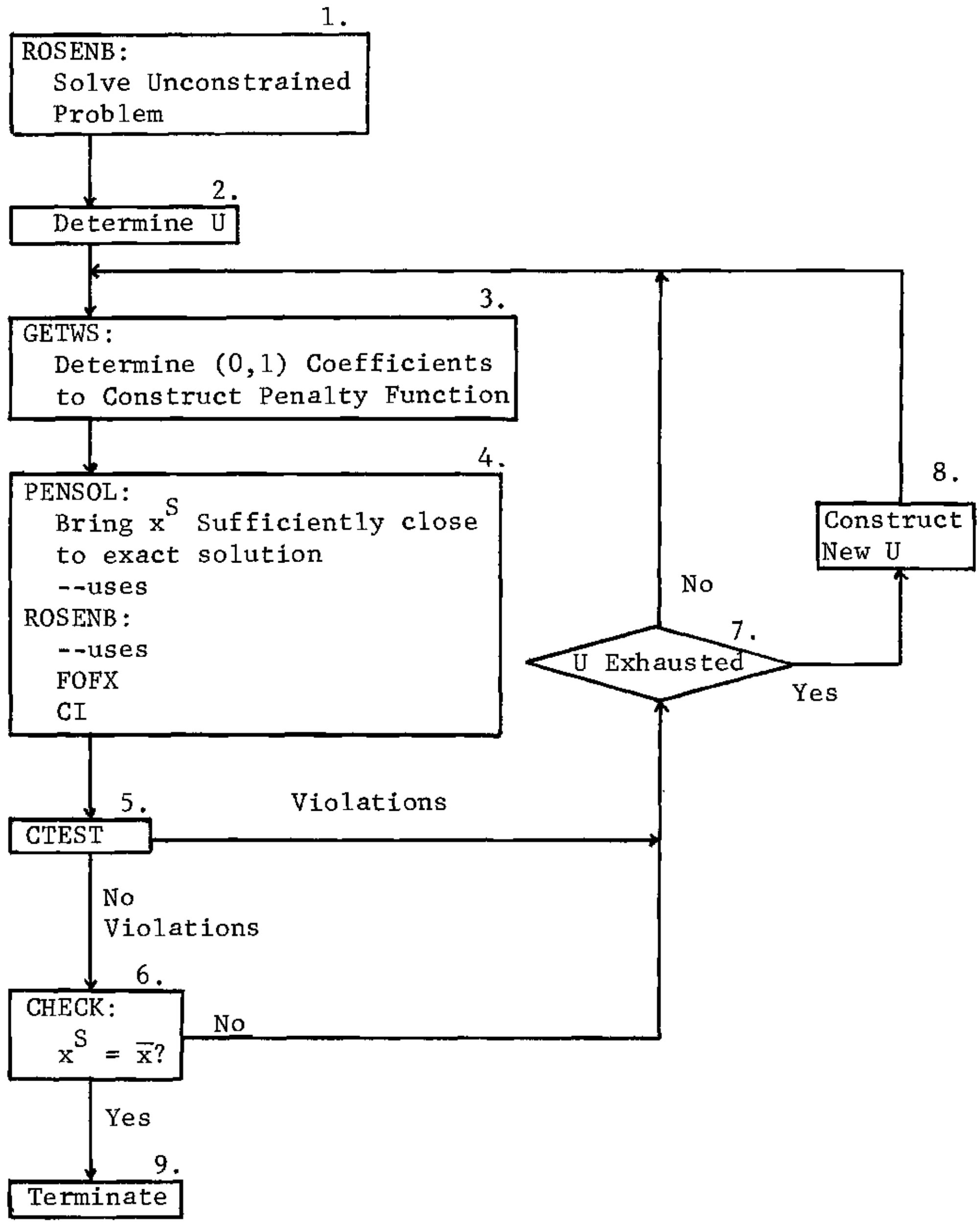

Figure 3. Solution Procedure

violates constraints 2, 4, and 6. At the first iteration we will have a MOLD matrix of the form

$$\text{MOLD} = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 0 & 0 \\ 6 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

indicating that we now will solve three subproblems, one for each row of the MOLD matrix. For example, in the first subproblem, constraint 2 alone will be included in the penalty function.

To select a single constraint to appear in the penalty function, a (0,1) coefficient, W(I), is used in the subroutines GETWS and FOFX. In GETWS, the appropriate row of the MOLD matrix is taken and the W(I) which corresponds to the constraint indices in that row are set equal to one. All others are set to zero. When the penalty function (4.1) is later evaluated during the search, only those terms with a nonzero W(I) coefficient will be included. Thus, only those constraints identified for that particular trial set by the MOLD matrix will be included.

Once these (0,1) coefficients have been determined, the penalty function parameters are initialized and subroutine PENSOL (Powell's penalty function method) is called to drive the solution to within CUTOF of the exact solution. It is this subroutine that is the heart of the solution procedure, controlling the convergence and calling the search routine. A flow chart of PENSOL appears in Figure 4. Some deviations from Powell's method occur in the execution of Block 2 where start points

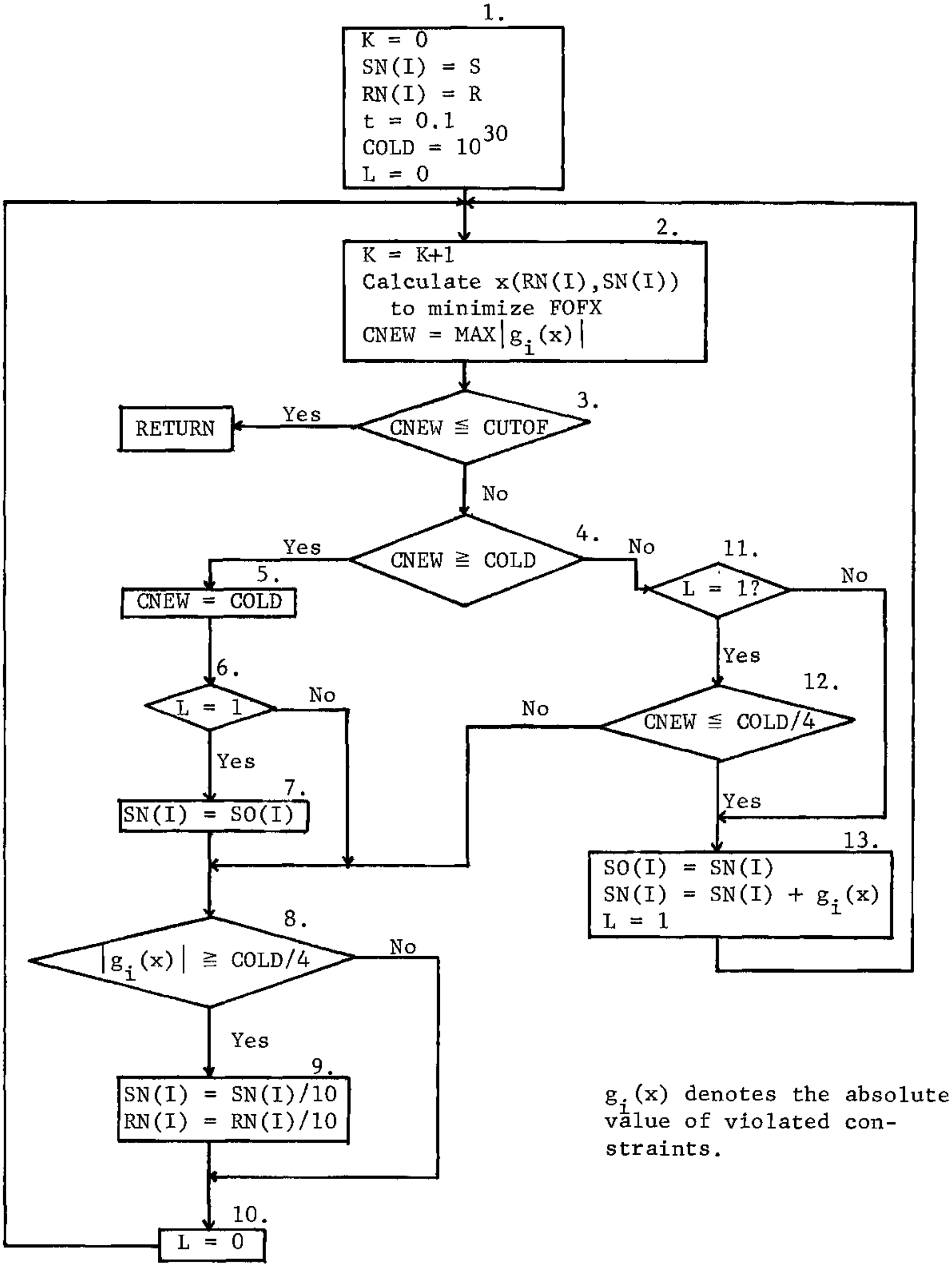

Figure 4. Powell's Penalty Function Method

for the search are determined and the Rosenbrock search is called. For the first iteration, the unconstrained solution is used as a start point. Thereafter, the solutions from the first iteration subproblems are used for subsequent start points for those problems which have the same first element of their MOLD row. That is, a problem holding constraints 2 and 5 as equalities used as its start point the solution to the subproblem that held 2 alone as an equality. A second difference occurs in the solution for x^S in Block 2. If the search detects that the current penalty function is unbounded, the search is halted when FOFX = 10^{20} and the corresponding solution vector saved to test for further constraint violations. The unbounded subproblem is then abandoned and the next subproblem considered.

Block 3 is where the different rules were inserted to control the search. The first rule is that shown where the search is discontinued when the maximum constraint violation, $g_i(x)$, is within some δ distance of the exact optimal for the subproblem in question. Thus, by controlling the value of CUTOF, we are able to control the exactness with which each subproblem is solved. It is CUTOF that was varied to determine the effects of relaxing the exactness of each solution.

In Block 4 we test for convergence. If the procedure is converging satisfactorily (i.e. the maximum violation is decreasing), we reduce parameter SN(I) (Block 13), making it more negative by the relation

$$SN(I) = SN(I) + CI(I)$$

where the CI(I) are the evaluations of the violated constraints only and therefore less than zero. This counters the decrease in magnitude of CI(I) which is a result of convergence, thus maintaining the effectiveness of the penalty term. If, on the other hand, we are not converging at a suitable rate, both parameters RN(I) and SN(I) (Block 9) corresponding to those constraints converging too slowly, are decreased by the same factor of 10 recommended by Powell (29). This has the effect of increasing the magnitude of the penalty term, giving more weight to the binding constraints in an effort to move the search closer to the point where they are satisfied as equalities.

Once we have driven the subproblem to within δ of the exact solution, we have an x^S which we are ready to test for constraint violations. The subroutine CTEST (Block 5, Fig. 3) calls the function subroutine CI(I) to evaluate the constraints, and if a violation occurs, the appropriate (0,1) entry is made in the VIOLAT matrix. The rows of VIOLAT correspond to the various trial sets of a particular iteration, and the columns correspond to the M constraints. Again, an entry of one indicates a violation and zero is entered otherwise. To illustrate, consider once again the hypothetical problem considered above. Suppose the first subproblem of the first iteration is the current trial set. Solving for x^S where S = 2, we find that constraints 1, 4, and 5 are violated. The resulting MOLD and VIOLAT matrices at this point are

$$\text{MOLD} = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 0 & 0 \\ 6 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad \text{VIOLAT} = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Encountering violations and seeing that U is not exhausted, the next subproblem with S = 4 is considered. Assume now that this second subproblem has been solved to within δ of the exact solution and when CTEST is called no violations are detected. The VIOLAT array would remain as shown above and this time we have a suspected optimal. CUTOF is reduced by

$$CUTOF = CUTOF/100$$

in this case and the search resumed to give a more exact solution. Since we suspect we are near the optimal solution, the iterations of ROSENB are also reduced by a factor of 2. Should this further search produce an x^S which still causes no violations, we are ready to check for optimality via Theil and Van de Panne's Rule 3.

Rule 3 is executed in the subroutine CHECK (Block 6, Fig. 3). CUTOF is again reduced by a factor of 100 and the iterations of ROSENB by another factor of 2. Constraints are removed one at a time from S leaving S-h, and the search resumed. When this new search is within CUTOF of its exact solution, a check is made to see if the h^{th} constraint is violated. If one or more of the h^{th} constraints is not violated, we have failed to find the true optimal, the CUTOF and ITRMAX are restored to their original values and the next subproblem is considered. Should we find each h^{th} constraint violated in this check, we have found the optimal solution.

Block 7, Fig. 3 tests U for exhaustion. We have discussed what happens when we enter this block and U is <u>not</u> exhausted and will now briefly explain the steps taken at the point when U <u>is</u> exhausted. When this situation arises, the MAIN program reconstructs a new MOLD corresponding to a new U for the next generation of trial sets (Block 8). A dummy matrix MNEW is formed which, one row at a time, copies the K entries of that row from MOLD each time a one is encountered in the corresponding row of the VIOLAT matrix and then adds the constraint index of the newly encountered violation in the $(k+1)^{st}$ column of MNEW. Consider the above example at the point where the first iteration has been completed and the optimal has not been found. Suppose the corresponding MOLD and VIOLAT matrices are

$$\text{MOLD} = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 0 & 0 \\ 6 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad \text{VIOLAT} = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

indicating that, where S = 2, x^S violates constraints 1, 4, and 5. With S = 4, constraint 5 is violated and for S = 6, 1 and 3 are violated. The MNEW matrix formed will be

$$\text{MNEW} = \begin{bmatrix} 2 & 1 & 0 & 0 & 0 & 0 \\ 2 & 4 & 0 & 0 & 0 & 0 \\ 2 & 5 & 0 & 0 & 0 & 0 \\ 4 & 5 & 0 & 0 & 0 & 0 \\ 6 & 1 & 0 & 0 & 0 & 0 \\ 6 & 3 & 0 & 0 & 0 & 0 \end{bmatrix}$$

MNEW is now relabeled as MOLD and we are ready to start the second iteration with a new U containing six elements rather than the three of the first iteration.

CHAPTER V

COMPUTATIONAL RESULTS

As mentioned in the introduction, the objectives of this research were essentially threefold. Firstly, we wanted to investigate computationally whether the Theil and Van de Panne algorithm could be used on nonlinear constraints. We also wished to investigate means of taking advantage of only approximating the optimal solution at each subproblem rather than solving it exactly, and wanted to use the approximated optimal solution to one subproblem as the starting point for the next subproblem. In this chapter, we will discuss the results of our experimentation and the successes and failures encountered in the pursuit of our objectives.

Recall that, in Geoffrion's extension to the combinatorial approach, he included all nonlinear constraints in the set X and addressed only linear constraints in a combinatorial manner. In this study, it was decided to treat all constraints in the combinatorial manner. Test problems with nonlinear constraints posed no computational difficulty even though no complete theoretical proofs are available for their convergence to the optimal. In this connection, the reader may refer to (10) where the proof of convergence for the general case contains an error. There is some reason to believe that the approach is still theoretically valid as borne out by the computational results here.

The five test problems used in this study are listed in Appendix A and were discussed in Chapter IV. The results of analysis are presented in Tables 1 and 2. To investigate the effect of relaxing the exactness with which each subproblem had to be solved, the five different rules discussed in Chapter IV were tested at different levels of exactness (values of δ). This was done for each of the four problems, P-1 through P-4. As would be expected, relaxing the exactness (increasing the value of CUTOF in the program) with which each subproblem is solved leads to reduced execution times (Table 1). As shown in Table 2, this was achieved with no appreciable loss in accuracy of results. It appears further that varying the decision rules had no effect on accuracy, but only on execution times.

In test problem P-1, all five exactness rules reacted similarly when exactness was relaxed (Figure 5). Rule 1, the greatest violation driven to less than δ, recommended by Powell when he introduced the penalty function used, was most efficient, taking less time for execution than the other rules by more than one second at CUTOF = .001 and .1 and nearly one second at CUTOF = .01. The fifth rule, using function evaluation improvement as a criterion for stopping the search, was least efficient by far, even when the exactness was relaxed beyond the other rules by a factor of 10. When relaxed by a factor of 100, rule 5 finally took less time than the fourth rule at its strictest CUTOF value. As the CUTOF value was increased, execution times for rules 2 and 4 decreased most rapidly, as might be expected, since they are dependent upon summations. This suggests that, if one continued to relax the exactness, these rules might prove to

Table 1. Execution Times

Decision Rule	δ CUTOF	Problem # P-1 Icount	P-1 Execution Time (sec)	P-2 Icount	P-2 Execution Time (sec)	P-3 Icount	P-3 Execution Time (sec)	P-4 Icount	P-4 Execution Time (sec)
1.	.001	90	15.548	44	9.837	11	2.371	7	1.818
	.01	74	13.997	41	9.174	7	1.844	8	1.660
	.1	61	10.391	35	7.305	5	1.394	7	1.536
	1.0	48	8.322	25	4.736	4	1.126		*
2.	.001	79	17.272	84	19.723	11	2.484	7	1.807
	.01	80	14.679	47	8.676	7	1.715	8	1.848
	.1	59	11.857	33	6.689	5	1.365	7	1.518
	1.0	45	7.586	25	5.229	4	1.090		*
3.	.001	102	19.018	65	12.473	8	2.208	10	2.227
	.01	89	16.622	41	9.598	7	1.747	9	2.230
	.1								
	1.0								
4.	.001	104	21.010	67	13.068	8	2.101	10	2.458
	.01	89	16.894	45	10.433	8	1.982	9	2.061
	.1								
	1.0								
5.	.001	129	28.619		*	9	2.277	21	4.226
	.01	112	25.005	127	28.681	8	2.149	21	4.234
	.1	121	23.630	73	11.740		†	21	4.035
	1.0	90	18.638	39	8.381		†	10	2.055

*Problem would not solve in alloted time. †No feasible solution indicated--see page 55.

Table 2. Functional Value at Solution Point

Decision Rule	δ CUTOF	Problem # P-1	P-2	P-3	P-4
1.	.001	99.99978	-44.00000	.1111121	-12.58607
	.01	99.99919	-44.00016	.1110956	-12.58634
	.1	100.0011	-44.00059	.1109440	-12.58669
	1.0	100.4059	-44.00201	.1089062	*
2.	.001	99.99996	-44.00215	.1111121	-12.58607
	.01	99.99915	-44.00006	.1110956	-12.58634
	.1	99.99969	-45.08987	.1109440	-12.58669
	1.0	100.0243	-44.00262	.1089062	*
3.	.001	99.99983	-44.00001	.1111155	-12.58612
	.01	99.99953	-44.00016	.1110920	-12.58612
4.	.001	99.99983	-44.00001	.1111155	-12.58612
	.01	99.99953	-44.00009	.1111155	-12.58612
5.	.001	100.0000	*	.1111155	-12.58608
	.01	100.0000	-44.00003	.1111083	-12.58607
	.1	99.99999	-44.00040	†	-12.58607
	1.0	100.0019	-44.00058	†	*

*Problem would not solve in alloted time.

†No feasible solution indicated--see page 55.

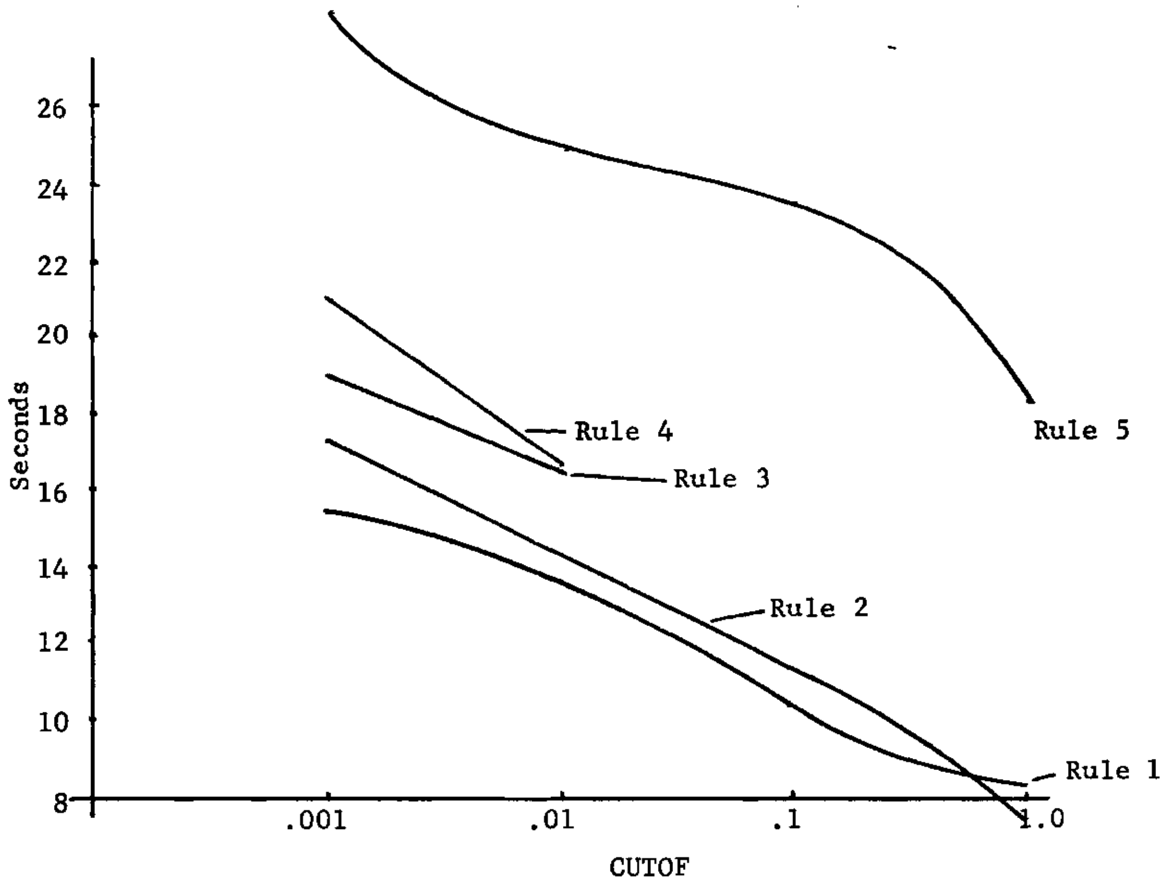

Figure 5. Problem P-1

be most efficient for this problem, and, at CUTOF = 1.0, we notice that rule 2 becomes more efficient than all others.

In test problem P-2 (Figure 6) the second rule, the sum of violations being driven to less than δ, was least efficient with CUTOF = .001 but most efficient with CUTOF = .01 and .10. As the exactness was relaxed further, rule 1 became the most efficient; however, the $\bar{x}$ vector solution obtained for rule 1 is not as exact as that obtained by rule 5 starting with the third decimal place. Should no greater accuracy be required, rule 1 would be the most desirable. Throughout, the $\bar{x}$ vectors agree out to three or four decimal places, indicating that the exactness could be relaxed even further and still maintain a fair degree of accuracy giving shorter execution times.

Test problem P-3 showed little response to change in CUTOF (Figure 7). At all CUTOF values all rules were within one second of each other in execution time. In this problem, rule 2 held both extremes, fastest with CUTOF = .01, .10, and 1.0 and slowest with CUTOF = .001. As indicated in Tables 1 and 2, this problem did not solve for δ = .1 and 1 using rule 5. At these values the change in functional evaluation is so slight that the search for optimum is halted before all constraints are driven to equalities. The solution procedure therefore passes $\bar{S}$ and reports no feasible solution, indicating that we have exceeded the level to which exactness can be relaxed for this problem.

In problem P-4, rule 1 proved again to be most efficient overall, and rule 5 the least efficient (Figure 8). All rules proved to be only slightly sensitive to changes in CUTOF.

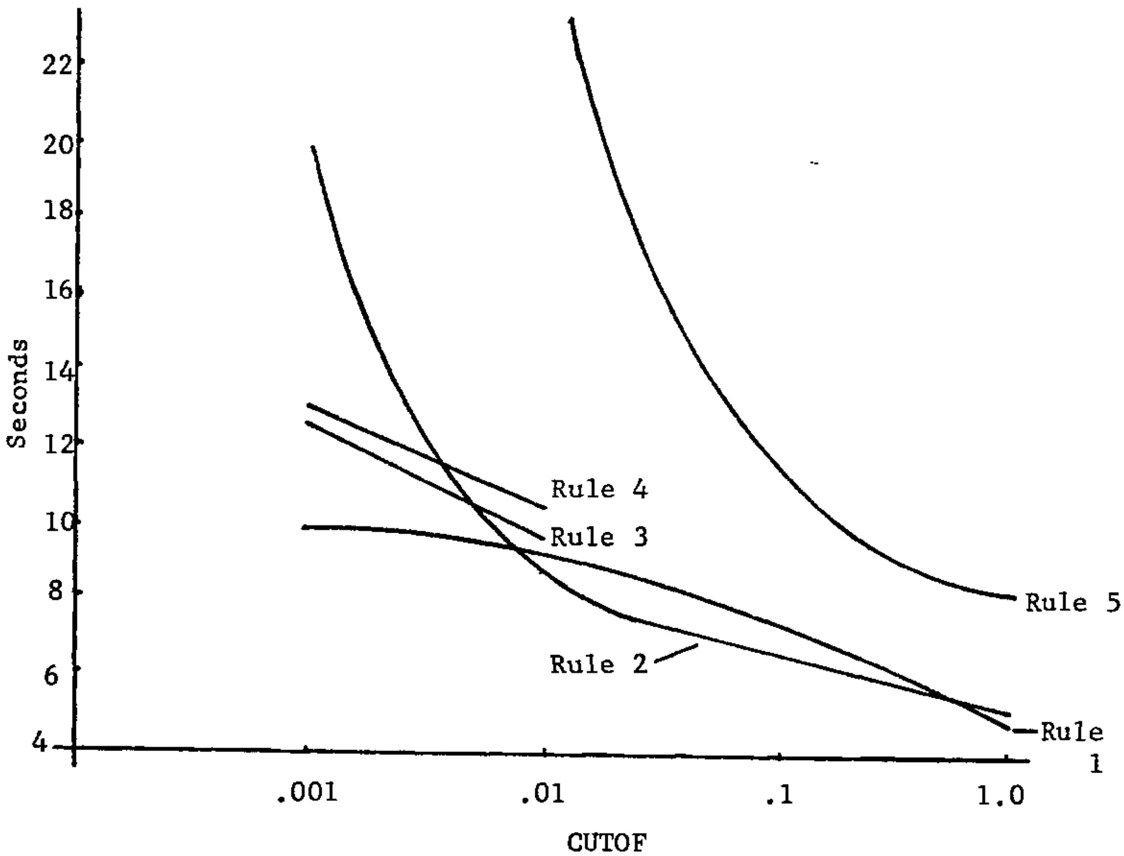

Figure 6. Problem P-2

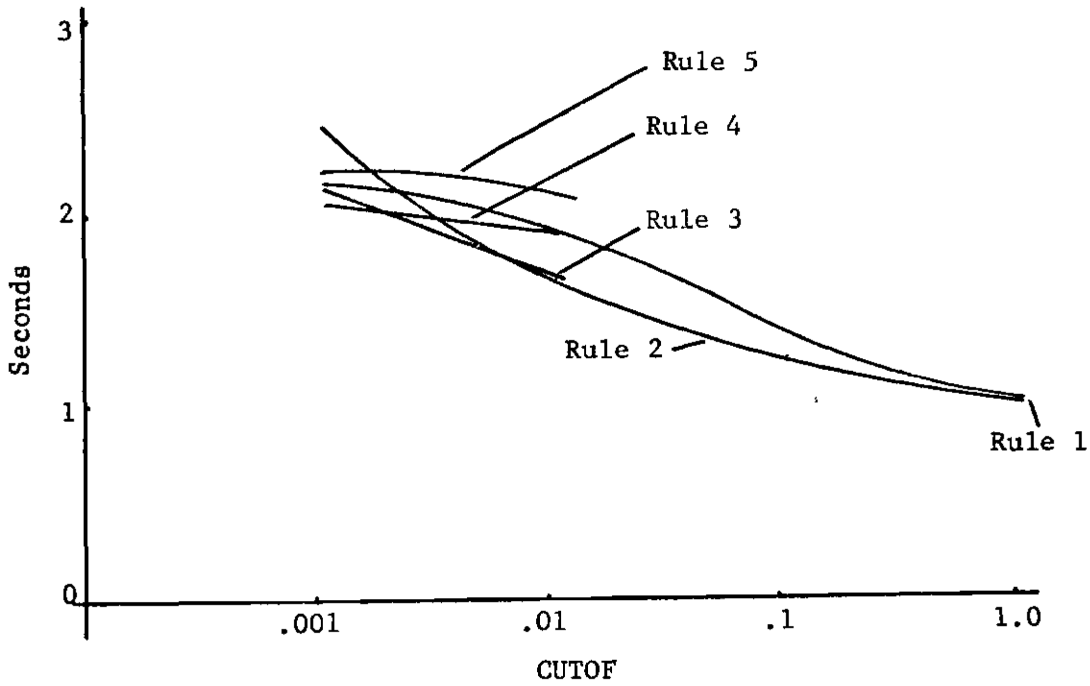

Figure 7. Problem P-3

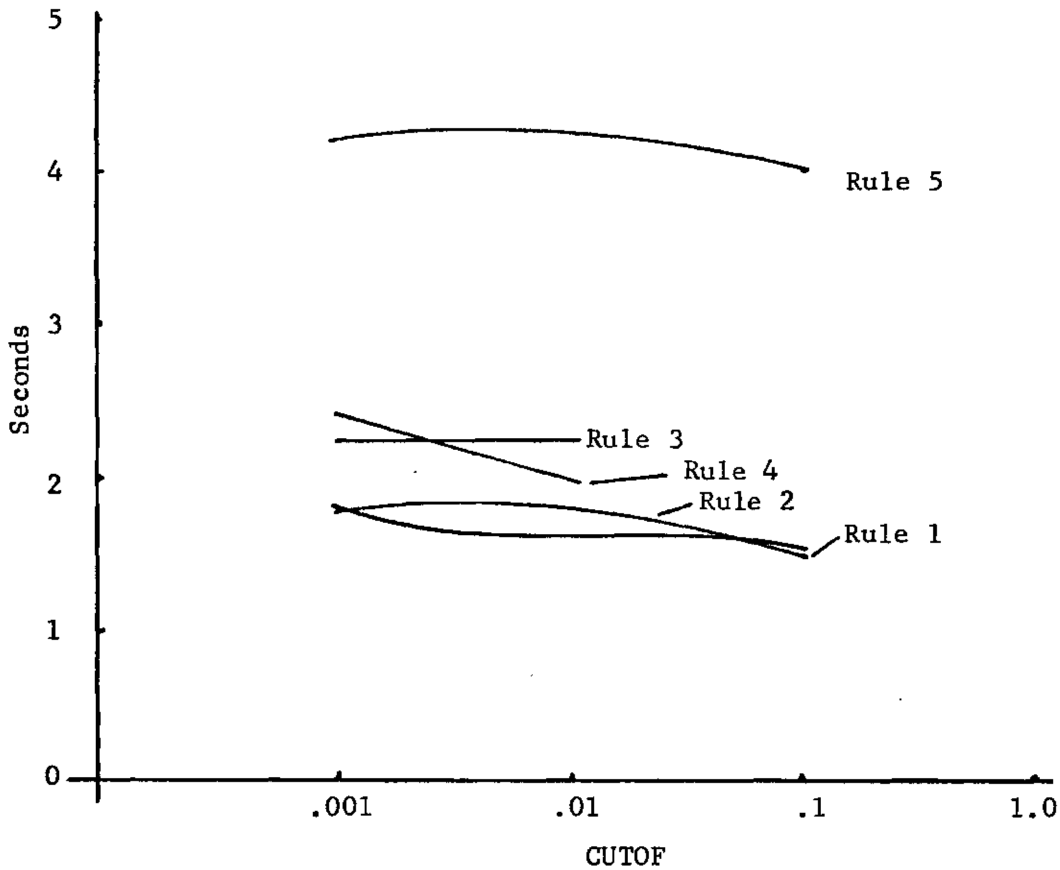

Figure 8. Problem P-4

In general, it was noticed that the fewer elements in $\overline{S}$, the less the effect of relaxing CUTOF. It was also noticed that the accuracy of the solution, $\bar{x}$, was reduced with the relaxation of CUTOF. Execution time was used as a measure of effectiveness for the rules used on each problem. A second statistic that can be used for the same purpose is ICOUNT, the number of times that the search subroutine, ROSENB, is called. This gives the number of x vector solutions, x^S, tested for optimality. A glance at Table 1 shows that these values correlate very closely with the execution times, but they were not used as a basis of comparison between decision rules since the program differed slightly from one rule to the next and the same number of calls on the search routine may take longer in one rule than in another. Trends are more apparent when using execution times, indicating more precisely which rules benefit most from relaxing the exactness with which the problems are solved.

One of the objectives of the study was to investigate how to take advantage of the closeness between various subproblems. It will be recalled that two successive subproblems derived from the same generation of trial sets may differ substantially from each other. In fact, the solution of one may not be an exterior point to the other, which is critical from the standpoint of the penalty function used. However, it may also be recalled that a problem in one generation of trial sets was derived from a previous generation (lineal predecessor) by adding a constraint as given by equation 2.2. Hence it is reasonable to expect that starting from the optimum of the lineal predecessor would be helpful. Besides, this start point is exterior to the new problem as desired.

The Problem of Convexity

For nonconvex problems where the unconstrained solution may be unbounded, recall from Chapter II that Geoffrion (19) recommends setting T_S equal to the indices of the constraints violated by some sequence $<x^v>$ feasible in P_S for which $<f(x^v)> \to \infty$. While this is not directly applicable to our solution procedure since we use an exterior penalty function method and remain outside the feasible space, similar steps were attempted in this study.

Test problem P-5 is an example of this situation, as the unconstrained problem is unbounded. The contours in the (x_1,x_3) plane of this problem are shown in Figure 9. Difficulties one might encounter in such a case are as follows.

When the search for the solution of the unconstrained problem or the unconstrained penalty function of a subproblem is cutoff at some preset bound due to the unboundedness of the problem, those constraints at the cutoff point were used to define T_S for the succeeding generation of trial sets. It was also necessary to insure that the start point to the succeeding subproblems was moved away from the cutoff point since, due to the nature of the Rosenbrock unconstrained search technique, a start point at the bound will cause the search to be cutoff again immediately and the next subproblem to be called. Any rule which will move the start point away from this bound will suffice, as long as the new start point found is still exterior to the subproblem being considered. In this study the arbitrary rule of dividing the unbounded point by factors of 100 and 1000 were tried successfully.

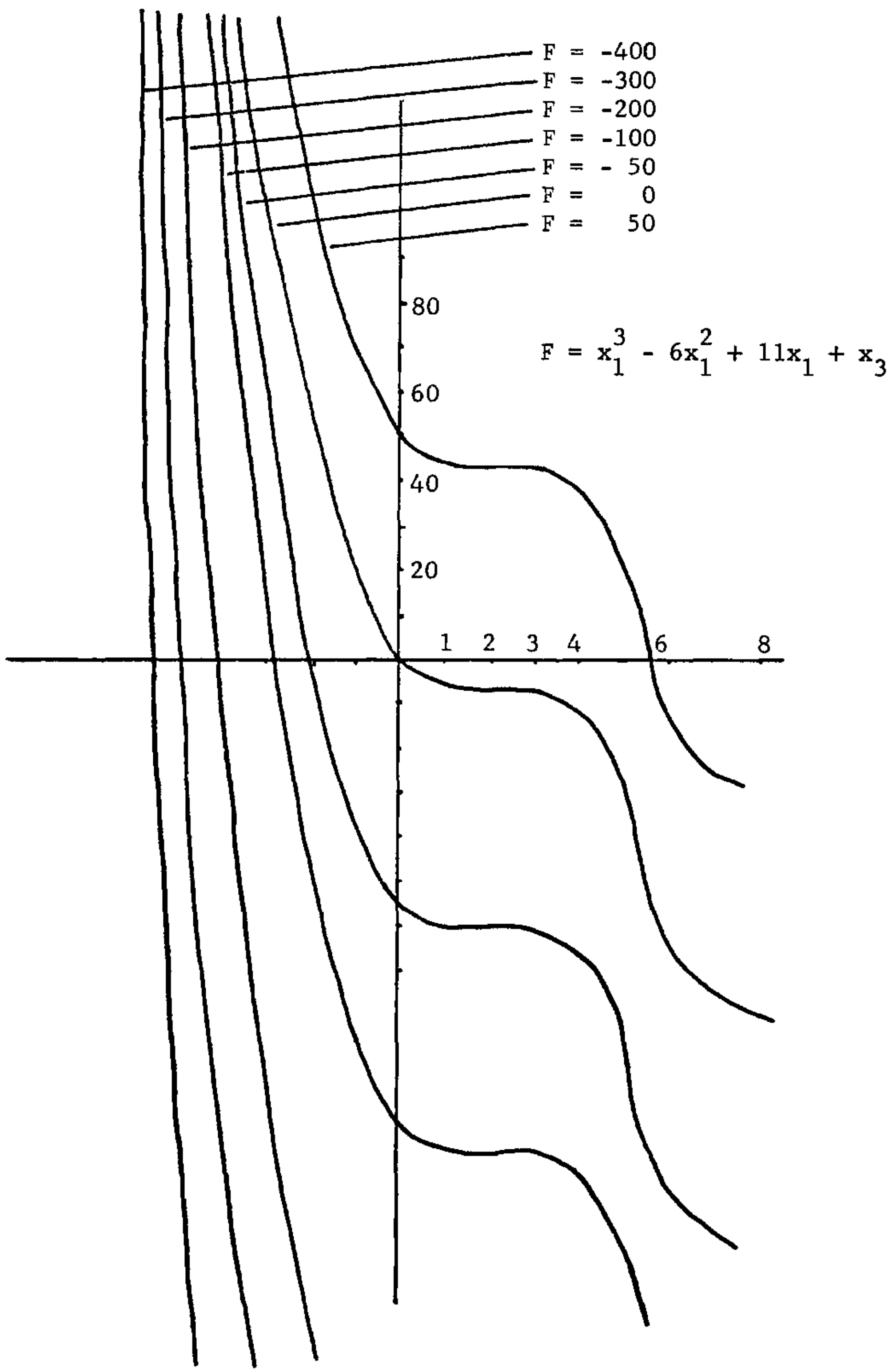

Figure 9. Contours of Test Problem P-5

In nonconvex problems the setting of the parameters r and s in the penalty function is also critical as arbitrary setting of r and s may not prevent the search from proceeding without bound as was the case in problem P-5. Sasson (34) recommends the following rules for setting r and s. Initialize r by: $r_i = g_i(x)/f(x)$ and initialize s by: $s_i = 0$, $i = 1,\ldots,m$. Use of these rules kept the search in problem P-5 from proceeding without bound as it did when the parameters were arbitrarily set.

The combinatorial approach and its extension address only problems where the constraint set is convex. The constraint set of problem P-5 is nonconvex. Attempts to solve this problem were unsuccessful until the cause of the nonconvexity of the constraint set was removed. When the problem was redefined without constraint 2, it solved with no difficulty.

CHAPTER VI

CONCLUSIONS AND RECOMMENDATIONS

In attempting to determine if the combinatorial approach could be applied to higher than second order functions, it was found that, for the convex functions tested with convex constraints, it can be applied without any difficulty. It was also found that nonlinear convex constraints could be included in the combinatorial treatment of constraints, although no theoretical proofs were found for the convergence of such a problem. This does, however, imply a hueristic notion that the combinatorial approach may be more general than suggested by Geoffrion (19).

Five different rules for terminating the optimization search were tested and it was found that, in each rule, as the exactness of the solution of each subproblem was relaxed, the execution time for the entire problem decreased, sometimes with no loss in accuracy. This effect is magnified as the number of constraints in $\overline{S}$ is increased.

Using the optimal of a lineal predecessor to a subproblem as the new start point also proved useful. This approach insured that the search for the solution to each subproblem began from an exterior point which is essential when an exterior penalty function method is used for solving the sequence of subproblems.

Test problems made clear the difficulties encountered when nonconvex constraints and/or nonconvex functions are addressed. No sure means of solving such problems was found; however, a greater understanding of the subject of convexity was gained through the attempts to solve them.

It is recommended that further investigation be made into the solution of nonconvex problems. Geoffrion recommends that the unbounded problem might be handled by setting T_S ($T_S \equiv \{i \in M\text{-}S : a_i x^S + b_i < 0\}$) equal to the indices of constraints that are violated by any sequence $\langle x^v \rangle$ feasible in (P_S) for which $\langle f(x^v) \rangle \to \infty$. In the solution procedure used, $\langle x^v \rangle$ is not feasible, but exterior to the feasible space, and it is possible that future research could pursue a means of bringing the unbounded solution (obtained when the search is artifically cut off) to be feasible in (P_S), perhaps by adjusting the parameters in the penalty function at the point where the search is halted.

It is also recommended that another means of defining succeeding generations of trial sets be investigated. Although the Lagrangian multiplier was not available in the solution to the numerical methods used in this study, it seems reasonable that its sign, which is the primary interest, might be determined for the problem

$$\text{Maximize: } f(x)$$

$$\text{Subject to: } g_i(x) \geqq b_i$$

by using the relation

$$\lambda_i = \partial f / \partial b_i$$

If b_i were perturbed so as to relax $g_i(x)$ slightly, the resulting change in f would indicate the sign of λ_i. This would permit the use of Geoffrion's method of updating the generation of trial sets, allowing S^o to be other than $\emptyset$. Further investigation of this approach might lead to an efficient means of addressing the constrained nonlinear programming problem.

APPENDIX A

TEST PROBLEMS

P-1 Maximize $f(x_1,x_2) = 10x_1 + 25x_2 - 10x_1^2 - x_2^2 - 4x_1x_2$

Subject to:
1. $x_1 + x_2 - 9 \geqq 0$
2. $x_1 + 2x_2 - 10 \geqq 0$
3. $x_1 \geqq 0$
4. $x_2 \geqq 0$

Start point: (1,1)

Solution point: $\bar{x} = (0,5)$ $f(\bar{x}) = 100$

Binding Constraints: 2 and 3

Source: Gue and Thomas (22)

P-2 Minimize: $f(x_1,x_2,x_3,x_4) = x_1^2 + x_2^2 + 2x_3^2 + x_4^2 - 5x_1 - 5x_2 - 21x_3 + 7x_4$

Subject to:
1. $-x_1^2 - x_2^2 - x_3^2 - x_4^2 - x_1 + x_2 - x_3 + x_4 + 8 \geqq 0$
2. $-x_1^2 - 2x_2^2 - x_3^2 - 2x_4^2 + x_1 + x_4 + 10 \geqq 0$
3. $-2x_1^2 - x_2^2 - x_3^2 - 2x_1 + x_2 + x_4 + 5 \geqq 0$

Start point: (0,0,0,0)

Solution point: $\bar{x} = (0,1,2,-1)$ $f(\bar{x}) = -44$

Binding Constraints: 1 and 3

Source: Kowalik and Osborn (24) but originally due to Rosen and Suzuki (32)

P-3 Minimize: $f(x_1,x_2,x_3) = 9 - 8x_1 - 6x_2 - 4x_3 + 2x_1^2 + 2x_2^2 + x_3^2 + 2x_1x_2 + 2x_1x_3$

Subject to:
1. $x_1 \geqq 0$
2. $x_2 \geqq 0$
3. $x_3 \geqq 0$
4. $-x_1 - x_2 - 2x_3 + 3 \geqq 0$

Start point: (1,1,1)

Solution point: $\bar{x} = (4/3, 7/9, 4/9)$ $f(\bar{x}) = 1/9$

Binding Constraints: 4

Source: E. M. L. Beale (1)

P-4 Minimize: $f(x_1,x_2) = x_1^2 + 3x_2^4 - 4x_2^3 - 12x_2^2$

Subject to:
1. $x_1 \geqq 0$
2. $x_2 \geqq 0$
3. $-x_1 - x_2 + 3 \geqq 0$
4. $-x_1^2 + 3x_1 - 4x_2 + 2 \geqq 0$
5. $-x_2 - 2.5 \geqq 0$

Start point: (1,1)

Solution point: $\bar{x} = (1.28, 1.05)$ $f(\bar{x}) = -12.58$

Binding Constraints: 4

Source: C. R. Swenson (37)

P-5 Minimize: $f(x_1,x_2,x_3) = x_1^3 - 6x_1^2 + 11x_1 + x_3$

Subject to: 1. $-x_1^2 - x_2^2 + x_3^2 \geqq 0$

2. $x_1^2 + x_2^2 + x_3^2 - 4 \geqq 0$

3. $-x_3 + 5 \geqq 0$

4. $x_1 \geqq 0$

5. $x_2 \geqq 0$

6. $x_3 \geqq 0$

Start point: (0,1,1)

Solution point: $\bar{x} = (0, \sqrt{2}, \sqrt{2})$

Binding Constraints: 1, 2, 4

Source: Fiacco and McCormick (15)

APPENDIX B

COMPUTER PROGRAM

```
@FOR,IS MAIN
FOR S9A-06/22-12:25 (,0)

   MAIN PROGRAM

   STORAGE USED: CODE(1) 000466; DATA(0) 000115; BLANK COMMON(2) 000000

    COMMON BLOCKS:

    0003    BLOKA   000150
    0004    BLOKB   064573
    0005    BLOKC   000004
    0006    BLOKD   000001
    0007    BLOKE   000013
    0010    BLOKF   000001
    0011    BLOKG   000024
    0012    BLOKH   000002

   EXTERNAL REFERENCES (BLOCK, NAME)

    0013    ROSENB
    0014    CI
    0015    GETWS
    0016    PENSOL
    0017    CTEST
    0020    CHECK
    0021    NINTR$
    0022    NRDU$
    0023    NIO1$
    0024    NIO2$
    0025    NWDU$
    0026    NSTOP$

   STORAGE ASSIGNMENT  (BLOCK, TYPE, RELATIVE LOCATION, NAME)

    0000    000012 1F         0001    000160 10L        0000    000050 12F        0001    000023 125G       0000    000056 13F
    0001    000031 132G       0001    000204 14L        0001    000062 145G       0000    000062 15F        0001    000102 157G
    0000    000076 16F        0001    000266 20L        0001    000146 200G       0001    000312 22L        0001    000221 231G
    0001    000342 24L        0001    000233 240G       0001    000234 242G       0001    000343 25L        0001    000246 251G
    0001    000255 255G       0001    000316 277G       0001    000433 29L        0000    000016 3F         0001    000364 320G
    0001    000373 323G       0001    000464 33L        0001    000401 330G       0001    000422 337G       0001    000444 352G
    0001    000456 356G       0001    000073 5L         0001    000113 6L         0001    000131 7L         0000    000024 8F
    0000    000036 9F         0011 R 000012 AX          0014 R 000000 CI          0004 R 064571 CUTOF       0004 R 064572 FO
    0000 I 000006 I           0006 I 000000 ICOUNT      0000 I 000007 IROW        0005    000000 ISTAGE     0005 I 000003 ISTGMX
    0010 I 000000 ISWIT       0005 I 000002 ITRMAX      0000 I 000010 J           0003 I 000147 K           0000 I 000011 L
    0005    000001 LCOUNT     0003 I 000145 M           0004 I 043120 MNEW        0004 I 021450 MOLD        0003 I 000144 N
    0007 I 000000 P           0000 I 000002 Q           0000 R 000004 R           0003 R 000050 RN          0000 R 000005 S
    0003 R 000106 SN          0012 R 000001 SUSMIN      0007 R 000001 SUSP        0000 I 000001 T           0011 R 000000 TEMPX
    0004 I 064570 TOTV        0012 R 000000 TRUVAL      0003 I 000146 U           0004 I 000000 VIOLAT      0003    000012 W
    0003 R 000000 X           0000 I 000000 Y           0000 I 000003 Z
```

```
00100     C
00100     C      FORTRAN  V FOR UNIVAC 1108
00100     C      CONSTRAINED NONLINEAR OPTIMIZATION
00100     C         WITH POWELL PENALTY FN AND ROSENBROCK SEARCH
00100     C
00101            COMMON/BLOK A/X(10),W(30),RN(30),SN(30),N,M,U,K
00103            COMMON/BLOK B/VIOLAT(300,30),MOLD(300,30),MNEW(300,30),TOTV,CUTOF,
00103           *             FO
00104            COMMON/BLOK C/ISTAGE,LCOUNT,ITRMAX,ISTGMX
00105            COMMON/BLOK D/ICOUNT
00106            COMMON/BLOK E/P,SUSP(10)
00107            COMMON/BLOK F/ISWIT
00110            COMMON/BLOK G/TEMPX(10),AX(10)
00111            COMMON/BLOK H/TRUVAL,SUSMIN
00112            INTEGER Y,U,T,P,Q,Z,TOTV,VIOLAT
00113          1 FORMAT (3F10.7,4I5,/(10F10.3))
00114            READ (5,1) CUTOF,R,S,N,M,ITRMAX,ISTGMX,(X(I),I=1,N)
00131            DO 2 I=1,N
00134          2 TEMPX(I)=X(I)
00134     C
00134     C      SOLVE UNCONSTRAINED PROB. USING ROSENBROCK
00134     C
00136            CALL ROSENB
00137            IF (ISWIT .NE. 1) GO TO 5
00141            WRITE (6,3)
00143          3 FORMAT (1X,28HUNCONSTRAINED SOL. UNBOUNDED)
00144            DO 4 I=1,N
00147            AX(I) = X(I)
00150          4 IF (ISWIT .EQ. 1) AX(I) = AX(I)/1000
00150     C
00150     C      USING UNCONSTRAINED SOLUTION DETERMINE WHICH
00150     C      CONSTRAINTS ARE VIOLATED AND FILL VIOLAT MATRIX
00150     C
00153          5 ICOUNT = 0
00154            IROW = 0
00155            K=0
00156            DO 7  I=1,M
00161             IF (CI(I) .LT. 0.0) GO TO 6
00163             VIOLAT(I,1)=0
00164            GO TO 7
00165          6 TOTV = TOTV+1
00166             IROW=IROW+1
00167             VIOLAT(IROW,I)=1
00170             MOLD(IROW,1)=I
00171          7 CONTINUE
00171     C
00171     C      IF NO CONST. ARE VIOLATED BY SOL. TO UNCONSTRAINED PROB,OPTIMAL
00171     C          OCCURS IN FEASIBLE SPACE
00171     C
00173            IF (TOTV .NE. 0) GO TO 10
00175            WRITE (6,8) FO,(X(Y),Y=1,N)
00204          8 FORMAT (14X,6HF(X) =,1PE17.6/17X,3HX =,1P6E17.6/
00204           1        (20X,1P6E17.6))
00205            WRITE (6,9)
```

```
00207      9 FORMAT (10X,50HUNCONSTRAINED OPTIMAL OCCURS WITHIN FEASIBLE SPACE)
00210        GO TO 33
00211     10 TOTV = 0.0
00212     11 K = K+1
00212  C
00212  C     IF ITERATION NUMBER EXCEEDS NO. OF CONSTRAINTS, NO FEASIBLE SOL.
00212  C        EXISTS
00212  C
00213        IF (K .LE. M) GO TO 14
00215        WRITE (6,12)
00217     12 FORMAT (10X,27HNO FEASIBLE SOLUTION EXISTS)
00220        WRITE (6,13) ICOUNT
00223     13 FORMAT (10X,8HICOUNT =,I5)
00224        GO TO 33
00225     14 WRITE (6,15) K,FO,(X(I),I=1,N)
00235     15 FORMAT (14X,3HK =,I2,5X,6HF(X) =,1PE17.6/5X,3HX =,1P6E17.6/
00235       *        (20X,1P6E17.6))
00236        WRITE (6,16) ((MOLD(I,J),J=1,10),I=1,IROW)
00247     16 FORMAT (10X,10I2)
00250     17 DO 25 U=1,IROW
00250  C
00250  C     DETERMINE THE PENALTY FUNCTION AND SOLVE FOR A NEW X VECTOR
00250  C
00253         CALL GETWS
00254        DO 18 I =1,M
00257        RN(I)=R
00260     18 SN(I)=S
00262     19  CALL PENSOL
00262  C
00262  C     DETERMINE IF NEW X VECTOR VIOLATES ANY CONSTRAINTS
00262  C
00263     20  CALL CTEST
00264     21  IF (TOTV .NE.   0) GO TO 24
00266         IF (P .NE. 0) GO TO 22
00266  C
00266  C     IF WE HAVE A SUSPECTED OPTIMAL, REDUCE THE CUTOF VALUE TO MOVE
00266  C        CLOSER TO THE EXACT OPTIMAL AND SEE IF OPTIMALITY TEST STILL
00266  C        SATISFIED
00266  C
00270        ITRMAX=(ITRMAX/2)
00271         CUTOF=(CUTOF/100)
00272         P=1
00273         CALL PENSOL
00274        ITRMAX=2*ITRMAX
00275         GO TO 20
00276     22 DO 23 I=1,N
00301     23 SUSP(I)=X(I)
00303        SUSMIN = TRUVAL
00304        ITRMAX=(ITRMAX/4)
00305         CALL CHECK
00306        IF (P .EQ. 868) GO TO 33
00310        ITRMAX=4*ITRMAX
00311         GO TO 25
00312     24  TOTV=0
```

```
 25 CONTINUE
 26 Z=IROW
    IROW=0.0
    DO 30  Q=1,Z
     DO 29 L=1,M
      IF (VIOLAT(Q,L) .EQ.   0) GO TO 29
    DO 27 T = 1,K
 27 IF (L .EQ. MOLD(Q,T)) GO TO 29
      IROW=IROW+1
      DO 28 J=1,K
 28    MNEW(IROW,J)=MOLD(Q,J)
      T=(K+1)
      MNEW(IROW,T)=L
 29  CONTINUE
 30 CONTINUE
    DO 32  I=1,IROW
    T=(K+1)
    DO 31  Y=1,T
 31   MOLD(I,Y)=MNEW(I,Y)
 32 CONTINUE
    GO TO 10
 33 CONTINUE
    END

END OF COMPILATION:        NO  DIAGNOSTICS.
```

```
@FOR,IS GETWS
FOR S9A-06/22-12:25 (,0)

   SUBROUTINE GETWS     ENTRY POINT 000050

   STORAGE USED: CODE(1) 000055; DATA(0) 000021; BLANK COMMON(2) 000000

    COMMON BLOCKS:

    0003    BLOKA   000150
    0004    BLOKB   064573

   EXTERNAL REFERENCES (BLOCK, NAME)

    0005    NWDUS
    0006    NI02$
    0007    NERR3$

   STORAGE ASSIGNMENT  (BLOCK, TYPE, RELATIVE LOCATION, NAME)

    0000    000002 1F        0001    000015 112G      0001    000030 122G      0001    000023 3L        0000 I 000000 B
    0004    064571 CUTOF     0004    064572 FO        0000    000007 INJP$     0000 I 000001 J         0003 I 000147 K
    0003 I  000145 M         0004    043120 MNEW      0004 I  021450 MOLD      0003    000144 N         0003   000050 RN
    0003    000106 SN        0004    064570 TOTV      0003 I  000146 U         0004    000000 VIOLAT    0003 I 000012 W
    0003    000000 X


00101             SUBROUTINE GETWS
00103             WRITE (6,1)
00105           1 FORMAT (1X,5HGETWS)
00105       C
00105       C     THE VALUS IN THE MOLD MATRIX IDENTIFY THE CONSTRAINTS THAT ARE TO
00105       C        BE DRIVEN TO EQUALITIES IN THE NEXT ITERATION
00105       C
00106             COMMON/BLOK A/X(10),W(30),RN(30),SN(30),N,M,U,K
00107             COMMON/BLOK B/VIOLAT(300,30),MOLD(300,30),MNEW(300,30),TOTV,CUTOF,
00107            *             FO
00110             INTEGER B,U,W
00111             DO 2  B=1,M
00114           2 W(B)=0
00116             IF(K .NE. 0) GO TO 3
00120             K=1
00121           3 DO 4  J=1,K
00124             B=MOLD(U,J)
00125           4 W(B)=1
00127             RETURN
00130             END
```

```
@FOR,IS PENSOL
FOR S9A-06/22-12:25 (,0)

   SUBROUTINE PENSOL    ENTRY POINT 000404

   STORAGE USED: CODE(1) 000415; DATA(0) 000653; BLANK COMMON(2) 000000

    COMMON BLOCKS:

    0003     BLOKA   000150
    0004     BLOKB   064573
    0005     BLOKD   000001
    0006     BLOKF   000001
    0007     BLOKG   000024

   EXTERNAL REFERENCES (BLOCK, NAME)

    0010     CI
    0011     ROSENB
    0012     NWDU$
    0013     NI02$
    0014     NI01$
    0015     NERR3$

   STORAGE ASSIGNMENT  (BLOCK, TYPE, RELATIVE LOCATION, NAME)

    0000     000557 1F        0001     000155 10L       0000     000570 11F       0001     000202 12L       0001     000025 123G
    0001     000207 13L       0001     000041 130G      0001     000225 14L       0001     000054 141G      0001     000227 15L
    0001     000065 150G      0000     000576 16F       0001     000111 164G      0001     000266 17L       0001     000271 18L
    0001     000273 19L       0000     000607 20F       0001     000161 207G      0001     000351 22L       0001     000356 23L
    0001     000221 232G      0000     000621 24F       0001     000233 242G      0001     000364 25L       0001     000366 26L
    0001     000321 271G      0001     000045 3L        0001     000337 302G      0001     000050 4L        0001     000060 6L
    0001     000100 7L        0000     000562 9F        0000 R 000552 ALARGE    0007 R 000012 AX        0000 R 000074 BX
    0010 R 000000 CI        0000 R 000000 CK        0000 R 000556 CNEW      0000 R 000553 COLD      0004 R 064571 CUTOF
    0004 R 064572 FO        0000 I 000554 I         0005 I 000000 ICOUNT    0000     000033 INJP$     0006 I 000000 ISWIT
    0000 I 000555 J         0003 I 000147 K         0000 I 000551 L         0003 I 000145 M         0004     043120 MNEW
    0004 I 021450 MOLD      0003 I 000144 N         0003 R 000050 RN        0003 R 000106 SN        0000 R 000036 SO
    0007     000000 TEMPX     0004     064570 TOTV      0003 I 000146 U         0004     000000 VIOLAT    0003 I 000012 W
    0003 R 000000 X         0000 I 000550 ZULU


00101      1*          SUBROUTINE PENSOL
00103      2*          WRITE (6,1)
00105      3*        1 FORMAT (1X,6HPENSOL)
00105      4*     C
00105      5*     C    PENSOL CONSTRUCTS THE POWELL PENALTY FUNCTION, VARYING THE
00105      6*     C           PARAMETERS AS NECESSARY
00105      7*     C
```

```
00106          COMMON/BLOK A/X(10),W(30),RN(30),SN(30),N,M,U,K
00107          COMMON/BLOK B/VIOLAT(300,30),MOLD(300,30),MNEW(300,30),TOTV,CUTOF,
00107         *              FO
00110          COMMON/BLOK D/ICOUNT
00111          COMMON/BLOK F/ISWIT
00112          COMMON/BLOK G/TEMPX(10),AX(10)
00113          DIMENSION CK(30),SO(30),BX(30,10)
00114          INTEGER W,U,ZULU
00115          L=0
00116          ALARGE = 10.0**30
00117          COLD=ALARGE
00120          IF (K .EQ. 1) GO TO 4
00122          DO 3 I=1,M
00125          IF (I .NE. MOLD(U,1)) GO TO 3
00127          DO 2 J=1,N
00132        2 X(J)=BX(I,J)
00134          GO TO 19
00135        3 CONTINUE
00137          GO TO 19
00140        4 DO 5 I = 1,N
00143        5 X(I)=AX(I)
00145          GO TO 19
00146        6 J = 0
00147          DO 7  I=1,M
00152          IF (W(I) .EQ. 0) GO TO 7
00154          J=J+1
00155          CK(J)=CI(I)
00156        7 CONTINUE
00160          CNEW=0.0
00161          IF (J .EQ. 0) GO TO 25
00163          DO 8  I=1,J
00166        8 IF (ABS(CK(I)) .GT. CNEW) CNEW=ABS(CK(I))
00171          WRITE (6,9)   CNEW,J
00175        9 FORMAT (14X,6HCNEW =,1PE17.10,5X,3HJ =,I2)
00176          IF (CNEW .LT. CUTOF) GO TO 26
00200          IF (CNEW .GE. COLD) GO TO 13
00202          IF (L .NE. 1) GO TO 10
00204          IF (CNEW .GT. COLD/4) GO TO 15
00204     C
00204     C    CONVERGING FAST ENOUGH - - REDUCT PARAMETER S ONLY
00204     C
00206       10 DO 12 I=1,M
00211          IF (W(I) .EQ. 0) GO TO 12
00213          SO(I)=SN(I)
00214          SN(I)=SN(I)+CI(I)
00215          WRITE (6,11)  SN(I),I
00221       11 FORMAT (10X,4HSN =,1PE17.10,5X,3HI =,I2)
00222       12 CONTINUE
00224          L=1
00225          GO TO 18
00226       13 CNEW=COLD
00227          IF (L .NE. 1) GO TO 15
00231          DO 14 I=1,M
```

```
00234              IF (W(I) .EQ. 0) GO TO 14
00236              SN(I)=SO(I)
00237           14 CONTINUE
00237        C
00237        C     NOT CONVERGING FAST ENOUGH - - REDUCE PARAMETERS R AND S
00237        C
00241           15 DO 17 I=1,M
00244              IF (W(I) .EQ. 0) GO TO 17
00246              IF (ABS(CI(I)) .LT. COLD/4) GO TO 17
00250              SN(I) = SN(I)/10
00251              RN(I) = RN(I)/10
00252              WRITE (6,16)  SN(I),RN(I),I
00257           16 FORMAT (10X,4HSN =,1PE17.10,5X,4HRN =,1PE17.10,5X,3HI =,I2)
00260           17 CONTINUE
00262              L=0
00263           18 COLD=CNEW
00263        C
00263        C     IF CURRENT PROB WAS UNBOUNDED, RESET START PT AND TRY NEXT SUBPROB
00263        C
00264           19 CALL ROSENB
00265              ICOUNT = ICOUNT+1
00266              WRITE (6,20)  FO,(X(I),I=1,N)
00275           20 FORMAT (14X,6HF(X) =,1PE17.6/17X,3HX= ,1P6E17.6/
00275             *        (20X,1P6E17.6))
00276              IF (K .NE. 1) GO TO 22
00300              ZULU = MOLD(U,1)
00301              DO 21  I=1,N
00304              BX(ZULU,I) = X(I)
00305           21 IF (ISWIT .EQ. 1) BX(ZULU,I) = BX(ZULU,I)/1000
00305        C
00305        C     IF CURRENT SUBPROB. UNBOUNDED, MOVE ON TO NEXT SUBPROB.
00305        C
00310           22 IF (ISWIT .EQ. 1) GO TO 23
00312              GO TO 6
00313           23 WRITE (6,24)
00315           24 FORMAT (1X,9HUNBOUNDED)
00316              GO TO 26
00317           25 CALL ROSENB
00320           26 RETURN
00321              END

       END OF COMPILATION:        NO  DIAGNOSTICS.
```

```
@FOR,IS CTEST
FOR S9A-06/22-12:25 (,0)

   SUBROUTINE CTEST     ENTRY POINT 000051

   STORAGE USED: CODE(1) 000055; DATA(0) 000015; BLANK COMMON(2) 000000

    COMMON BLOCKS:

    0003    BLOKA   000150
    0004    BLOKB   064573

   EXTERNAL REFERENCES (BLOCK, NAME)

    0005    CI
    0006    NWDU$
    0007    NI02$
    0010    NERR3$

   STORAGE ASSIGNMENT  (BLOCK, TYPE, RELATIVE LOCATION, NAME)

    0000    000001 1F        0001    000011 112G      0001    000026 2L        0001    000036 3L        0005 R 000000 CI
    0004 R 064571 CUTOF     0004    064572 FO        0000 I 000000 I         0000    000005 INJP$     0003    000147 K
    0003 I 000145 M         0004    043120 MNEW      0004    021450 MOLD      0003    000144 N         0003    000050 RN
    0003    000106 SN        0004 R 064570 TOTV      0003 I 000146 U         0004 R 000000 VIOLAT    0003    000012 W
    0003    000000 X


00101      1*           SUBROUTINE CTEST
00103      2*           WRITE (6,1)
00105      3*         1 FORMAT (1X,5HCTEST)
00105      4*      C
00105      5*      C    CTEST DETERMINES WHICH CONSTRAINTS ARE VIOLATED BY THE PRESENT
00105      6*      C        SOLUTION
00105      7*      C
00106      8*           COMMON/BLOK A/X(10),W(30),RN(30),SN(30),N,M,U,K
00107      9*           COMMON/BLOK B/VIOLAT(300,30),MOLD(300,30),MNEW(300,30),TOTV,CUTOF,
00107     10*          *             FO
00110     11*           INTEGER U
00111     12*           DO 3  I=1,M
00114     13*            IF (CI(I) .LT. (-CUTOF)) GO TO 2
00116     14*            VIOLAT(U,I)=0.0
00117     15*            GO TO 3
00120     16*         2  TOTV=TOTV+1
00121     17*            VIOLAT(U,I)=1
00122     18*         3 CONTINUE
00124     19*           RETURN
00125     20*           END
```

```
@FOR,IS CHECK
FOR S9A-06/22-12:25 (,0)

   SUBROUTINE CHECK      ENTRY POINT 000130

   STORAGE USED: CODE(1) 000137; DATA(0) 000052; BLANK COMMON(2) 000000

    COMMON BLOCKS:

    0003    BLOKA   000150
    0004    BLOKB   064573
    0005    BLOKD   000001
    0006    BLOKE   000013
    0007    BLOKH   000002

   EXTERNAL REFERENCES (BLOCK, NAME)

    0010    PENSOL
    0011    CI
    0012    NWDU$
    0013    NI02$
    0014    NI01$
    0015    NERR3$

   STORAGE ASSIGNMENT  (BLOCK, TYPE, RELATIVE LOCATION, NAME)

    0000    000003 1F       0001    000011 115G     0001    000024 120G     0001    000073 144G     0001    000033 3L
    0000    000005 5F       0000    000012 6F       0000    000024 7F       0001    000110 8L       0001    000114 9L
    0011 R  000000 CI       0004 R  064571 CUTOF    0000 I  000000 F        0004    064572 FO       0000 I  000001 G
    0005 I  000000 ICOUNT   0000    000034 INJP$    0003 I  000147 K        0003 I  000145 M        0004    043120 MNEW
    0004 I  021450 MOLD     0003 I  000144 N        0006 I  000000 P        0003    000050 RN       0003    000106 SN
    0007 R  000001 SUSMIN   0006 R  000001 SUSP     0004    064570 TOTV     0007    000000 TRUVAL   0003 I  000146 U
    0004    000000 VIOLAT   0003 I  000012 W        0003    000000 X        0000 I  000002 Y


00101               SUBROUTINE CHECK
00103               WRITE (6,1)
00105             1 FORMAT (1X,5HCHECK)
00105         C
00105         C     CHECK DETERMINES IF THE SUSPECTED OPTIMAL IS IN FACT THE TRUE OPT
00105         C
00106               COMMON/BLOK A/X(10),W(30),RN(30),SN(30),N,M,U,K
00107               COMMON/BLOK B/VIOLAT(300,30),MOLD(300,30),MNEW(300,30),TOTV,CUTOF,
00107             *              FO
00110               COMMON/BLOK D/ICOUNT
00111               COMMON/BLOK E/P,SUSP(10)
00112               COMMON/BLOK H/TRUVAL,SUSMIN
00113               INTEGER F,G,W,U,Y,P
```

```
00114          DO 4  F=1,K
00117           DO 2  G=1,M
00122            IF ((G-MOLD(U,F)) .EQ. 0) GO TO 3
00124        2  CONTINUE
00126        3  W(G)=0
00127          CUTOF = (CUTOF/100)
00130            CALL PENSOL
00131            W(G)=1
00132           IF (CI(G) .GE. (-CUTOF)) GO TO 8
00134        4 CONTINUE
00136          WRITE (6,5)
00140        5 FORMAT (10X,20HOPTIMAL SOLUTION IS:)
00141          WRITE (6,6) SUSMIN, (SUSP(Y),Y=1,N)
00150        6 FORMAT (14X,6HF(X) =,1PE17.6/17X,3HX =,1P6E17.6/
00150         1       (20X,1P6E17.6))
00151          WRITE (6,7)   ICOUNT
00154        7 FORMAT (10X,8HICOUNT =,I5)
00155          P=888
00156            GO TO 9
00157        8 P=0
00160          CUTOF = (CUTOF*10000)
00161        9 CONTINUE
00162          RETURN
00163          END

       END OF COMPILATION:        NO  DIAGNOSTICS.
```

```
@FOR,IS FOFX
FOR S9A-06/22-12:25 (,0)

   FUNCTION FOFX        ENTRY POINT 000060

   STORAGE USED: CODE(1) 000064; DATA(0) 000015; BLANK COMMON(2) 000000

    COMMON BLOCKS:

    0003    BLOKA   000150

   EXTERNAL REFERENCES (BLOCK, NAME)

    0004    CI
    0005    NERR3$

   STORAGE ASSIGNMENT  (BLOCK, TYPE, RELATIVE LOCATION, NAME)

    0001    000044 1L         0001    000026 110G       0004 R 000000 CI         0000 R 000000 FOFX       0000 I 000002 I
    0000    000006 INJP$      0003    000147 K          0003 I 000145 M          0003   000144 N          0003 R 000050 RN
    0003 R 000106 SN          0000 R 000001 TRUVAL      0003   000146 U          0003 I 000012 W          0003 R 000000 X


00101          FUNCTION FOFX(DUM)
00101     C
00101     C    FOFX EVALUATES THE PENALTY FUNCTION FOR THE CURRENT VALUES OF X(I)
00101     C
00101     C    IF A MINIMIZATION PROBLEM,
00101     C    ALTERNATE METHOD --- CHANGE .GE. TO .LE. AFTER COMMENT C9999.
00101     C
00103          COMMON/BLOK A/X(10),W(30),RN(30),SN(30),N,M,U,K
00104          INTEGER W
00105          FOFX=-10*(X(1))-25*(X(2))+(10*(X(1)**2))+((X(2))**2)+4*((X(1))*(X(
00105         *2)))
00106          TRUVAL = FOFX
00107          DO 1 I=1,M
00112          IF (W(I) .EQ.   0) GO TO 1
00114          FOFX=FOFX+W(I)*((CI(I)+SN(I))**2)/RN(I)
00115        1 CONTINUE
00117          RETURN
00120          END

      END OF COMPILATION:        NO  DIAGNOSTICS.
```

```
@FOR,IS CI
FOR S9A-06/22-12:25 (,0)

   FUNCTION CI           ENTRY POINT 000050

   STORAGE USED: CODE(1) 000054; DATA(0) 000012; BLANK COMMON(2) 000000

    COMMON BLOCKS:

    0003   BLOKA  000150

   EXTERNAL REFERENCES (BLOCK, NAME)

    0004   NERR2$
    0005   NERR3$

   STORAGE ASSIGNMENT  (BLOCK, TYPE, RELATIVE LOCATION, NAME)

    0001   000012 1L       0001   000021 2L       0001   000031 3L       0001   000036 4L       0000 R 000000 CI
    0000   000004 INJP$    0003   000147 K        0003   000145 M        0003   000144 N        0003   000050 RN
    0003   000106 SN       0003   000146 U        0003   000012 W        0003 R 000000 X


00101      1*          FUNCTION CI(I)
00101      2*    C
00101      3*    C     CI(I) EVALUATES THE CONSTRAINTS
00101      4*    C
00103      5*          COMMON/BLOK A/X(10),W(30),RN(30),SN(30),N,M,U,K
00104      6*          GO TO (1,2,3,4),I
00105      7*        1 CI=-X(1)-X(2)+9
00106      8*          RETURN
00107      9*        2 CI=-X(1)-2*(X(2))+10
00110     10*          RETURN
00111     11*        3 CI=X(1)
00112     12*          RETURN
00113     13*        4 CI=X(2)
00114     14*          RETURN
00115     15*          END

      END OF COMPILATION:        NO  DIAGNOSTICS.
```

```
@FOR,IS ROSENB
FOR S9A-06/22-12:25 (,0)

   SUBROUTINE ROSENB    ENTRY POINT 000621

   STORAGE USED: CODE(1) 000636; DATA(0) 000610; BLANK COMMON(2) 000000

    COMMON BLOCKS:

    0003    BLOKA  000150
    0004    BLOKB  064573
    0005    BLOKC  000004
    0006    BLOKF  000001

   EXTERNAL REFERENCES (BLOCK, NAME)

    0007    FOFX
    0010    LINES
    0011    BUMP
    0012    NWDU$
    0013    NI02$
    0014    SQRT
    0015    NSTOP$
    0016    NERR3$

   STORAGE ASSIGNMENT  (BLOCK, TYPE, RELATIVE LOCATION, NAME)

    0000    000534 1F        0001    000163 11L       0001    000222 13L       0001    000244 14L       0001    000104 143G
    0001    000105 146G      0001    000121 155G      0001    000305 16L       0001    000142 165G      0001    000315 17L
    0001    000201 201G      0001    000274 224G      0001    000345 244G      0001    000346 247G      0001    000360 255G
    0001    000374 260G      0001    000377 263G      0001    000424 274G      0001    000067 3L        0001    000561 30L
    0001    000437 302G      0001    000447 307G      0001    000565 31L       0001    000466 313G      0001    000470 317G
    0001    000570 32L       0001    000476 324G      0001    000577 33L       0001    000522 334G      0001    000534 342G
    0001    000553 352G      0001    000115 6L        0001    000127 8L        0001    000131 9L        0000 R  000454 A
    0000 R  000000 ALPHA     0000 R  000144 BETA      0004    064571 CUTOF     0000 R  000466 D         0000 R  000532 DOT
    0000 R  000521 DUM       0000 R  000500 E         0004 R  064572 FO        0007 R  000000 FOFX      0000 R  000525 F1
    0000 I  000524 I         0000 I  000523 IK        0000    000557 INJP$     0005 I  000000 ISTAGE    0005 I  000003 ISTGMX
    0006 I  000000 ISWIT     0000 I  000516 ITRIAL    0005 I  000002 ITRMAX    0000 I  000522 J         0003    000147 K
    0000 I  000526 L         0005 I  000001 LCOUNT    0003    000145 M         0000 I  000531 MMO       0004    043120 MNEW
    0004    021450 MOLD      0003 I  000144 N         0000 I  000515 NCASE     0000 I  000517 NL        0000 I  000520 NXTMAX
    0003    000050 RN        0003    000106 SN        0000 R  000513 STG       0000 R  000527 SUM       0000 R  000533 SUMRT
    0000 R  000530 SUMRT1    0004    064570 TOTV      0000 R  000514 TRI       0003    000146 U         0000 R  000310 V
    0004    000000 VIOLAT    0003 I  000012 W         0003 R  000000 X         0000 I  000512 Y


00101      1*          SUBROUTINE ROSENB
00103      2*          WRITE (6,1)
00105      3*        1 FORMAT (1X,6HROSENB)
```

```
00106        COMMON/BLOK A/X(10),W(30),RN(30),SN(30),N,M,U,K
00107        COMMON/BLOK B/VIOLAT(300,30),MOLD(300,30),MNEW(300,30),TOTV,CUTOF,
00107       *              FO
00110        COMMON/BLOK C/ISTAGE,LCOUNT,ITRMAX,ISTGMX
00111        COMMON/BLOK F/ISWIT
00112        DIMENSION ALPHA(10,10),BETA(10,10),V(10,10),A(10),D(10),E(10)
00113        INTEGER Y,W
00114        DATA STG/6HSTAGES/, TRI/6HTRIALS/                                  ROSE 045
00117        NCASE=0
00120      2 ITRIAL=0
00121        ISTAGE=0
00122        LCOUNT=0
00123        NCASE=NCASE+1
00124        NL=N+8
00125        IF (N .GT. 6) NL=2*N+9
00127        NXTMAX=75*N
00130        IF (ITRMAX .LT. 1) ITRMAX=50*N
00132        IF (ISTGMX .LT. 1) ISTGMX=25*N
00134        FO=FOFX(DUM)
00135        IF (ABS(FO) .LT. 10.0**20) GO TO 3
00137        ISWIT=1
00140        GO TO 33
00141      3 CALL LINES (NL)
00142        DO 5    J=1,N
00145        DO 4    IK=1,N
00150      4 V(J,IK)=0.0
00152      5 V(J,J)=1.0
00154      6 DO 7    J=1,N
00157        A(J)=2.0
00160        D(J)=0.0
00161      7 E(J)=0.1
00163      8 I=1
00164      9 DO 10   J=1,N
00167     10 X(J)=X(J)+E(I)*V(I,J)
00171        F1=FOFX(DUM)
00172        IF (ABS(F1) .LT. 10.0**20) GO TO 11
00174        ISWIT=1
00175        GO TO 33
00175  C
00175  C     FOR MIN PROB, CHANGE .GE. TO .LE. IN NEXT STATEMENT
00175  C
00176     11 IF (F1 .LE. FO) GO TO 13
00200        DO 12   Y=1,N
00203     12 X(Y)=X(Y)-E(I)*V(I,Y)
00205        E(I)=-.5*E(I)
00206        IF (A(I) .LT. 1.5) A(I)=0.0
00210        GO TO 14
00211     13 D(I)=D(I)+E(I)
00212        E(I)=3.*E(I)
00213        FO=F1
00214        IF (A(I) .GT. 1.5) A(I)=1.0
00216     14 ITRIAL=ITRIAL+1
00217        IF (ITRIAL .GT. ITRMAX) GO TO 30
00221        IF (NXTMAX .EQ. ITRIAL) CALL BUMP(X,N,NXTMAX,FO,E,D,W)
```

```
00223            DO 15   J=1,N
00226            IF (A(J) .GT. 0.5) GO TO 16
00230         15 CONTINUE
00232            GO TO 17
00233         16 IF (I .EQ. N) GO TO 8
00235            I=I+1
00236            GO TO 9
00237         17 ISTAGE=ISTAGE+1
00240            IF (ISTAGE .GT. ISTGMX) GO TO 31
00242            NXTMAX=ITRIAL+75*N
00243            DO 18   J=1,N
00246            DO 18   IK=1,N
00251         18 ALPHA(J,IK)=0.0
00254            DO 20   J=1,N
00257            DO 20   Y=1,N
00262            DO 19   L=J,N
00265         19 ALPHA(J,Y)=ALPHA(J,Y)+D(L)*V(L,Y)
00267         20 BETA(J,Y)=ALPHA(J,Y)
00272            SUM=0.0
00273            DO 21   Y=1,N
00276         21 SUM=SUM+BETA(1,Y)**2
00300            SUMRT1=SQRT(SUM)
00301            DO 22   Y=1,N
00304         22 V(1,Y)=BETA(1,Y)/SUMRT1
00306            DO 28   Y=2,N
00311            MMO=Y-1
00312            DO 25   J=1,MMO
00315            DOT=0.0
00316            DO 23   IK=1,N
00321         23 DOT=DOT+ALPHA(Y,IK)*V(J,IK)
00323            DO 24   IK=1,N
00326         24 BETA(Y,IK)=BETA(Y,IK)-DOT*V(J,IK)
00330         25 CONTINUE
00332            SUM=0.0
00333            DO 26   IK=1,N
00336         26 SUM=SUM+BETA(Y,IK)**2
00340            SUMRT=SQRT(SUM)
00341            DO 27   IK=1,N
00344         27 V(Y,IK)=BETA(Y,IK)/SUMRT
00346         28 CONTINUE
00350            SUM=0.0
00351            DO 29   IK=1,N
00354         29 SUM=SUM+ALPHA(2,IK)**2
00356            GO TO 6
00357         30 CALL LINES(NL)
00360            GO TO 32
00361         31 CALL LINES(NL)
00362         32 IF (NCASE .GT. 9) STOP
00364         33 CONTINUE
00365            RETURN
00366            END

      END OF COMPILATION:       NO  DIAGNOSTICS.
```

```
@FOR,IS BUMP
FOR S9A-06/22-12:25 (,0)

   SUBROUTINE BUMP      ENTRY POINT 000065

   STORAGE USED: CODE(1) 000111; DATA(0) 000030; BLANK COMMON(2) 000000

   EXTERNAL REFERENCES (BLOCK, NAME)

    0003   LINES
    0004   FOFX
    0005   NEXP1$
    0006   NERR3$

   STORAGE ASSIGNMENT  (BLOCK, TYPE, RELATIVE LOCATION, NAME)

    0001   000022 107G      0000 R 000001 DUM      0004 R 000000 FOFX      0000 I 000000 I       0000   000007 INJP$


00101      1*          SUBROUTINE BUMP(X,N,K,F,E,D,W)
00103      2*          DIMENSION E(10),D(10),X(10)
00104      3*          CALL LINES(2)
00105      4*          K=K+75*N
00106      5*          DO 1 I=1,N
00111      6*          D(I)=0.
00112      7*          E(I)=0.1
00113      8*        1 X(I)=X(I)+(X(I)/8.)*(-1)**I
00115      9*          F=FOFX(DUM)
00116     10*          RETURN
00117     11*          END

      END OF COMPILATION:       NO  DIAGNOSTICS.
```

```
@FOR,IS LINES
FOR S9A-06/22-12:25 (,0)

   SUBROUTINE LINES     ENTRY POINT 000024

   STORAGE USED: CODE(1) 000026; DATA(0) 000005; BLANK COMMON(2) 000000

    COMMON BLOCKS:

    0003   BLOKC  000004

   EXTERNAL REFERENCES (BLOCK, NAME)

    0004   NERR3$

   STORAGE ASSIGNMENT  (BLOCK, TYPE, RELATIVE LOCATION, NAME)

    0000    000000 INJP$     0003    000000 ISTAGE    0003    000003 ISTGMX    0003    000002 ITRMAX    0003 I 000001 LCOUNT


00101      1*          SUBROUTINE LINES(N)
00103      2*          COMMON/BLOK C/ISTAGE,LCOUNT,ITRMAX,ISTGMX
00104      3*          LCOUNT=LCOUNT+N
00105      4*          IF (LCOUNT .LT. 57) RETURN
00107      5*          LCOUNT=N+1
00110      6*          RETURN
00111      7*          END

      END OF COMPILATION:       NO  DIAGNOSTICS.
```

BIBLIOGRAPHY

1. Beale, E. M. L.; "On Quadratic Programming"; Naval Research Logistics Quarterly, V6, 227-243, 1959.

2. Beveridge, G. S., R. S. Schechter; Optimization: Theory and Practice; McGraw-Hill Book Company, New York, 1970.

3. Box, M. J.; "A New Method of Constrained Optimization and a Comparison with Other Methods"; Computer Journal, V8, 42, 1965.

4. __________; "A Comparison of Several Current Optimization Methods and The Use of Transformations in Constrained Problems"; Computer Journal, V9, 67, 1966.

5. Box, M. J., D. Davies, W. H. Swann; Nonlinear Optimization Techniques; Monograph #5; Oliver and Boyd Ltd, London, 1969.

6. Carroll, C. W.; "The Created Response Surface Technique for Optimizing Non-Linear Restrained Systems"; Operations Research, V9, 169-185, 1961.

7. Dantzig, G. B.; Linear Programming and Extensions; Princeton University Press, Princeton, N. J., 1963.

8. Davidon, W. C.; "Variable Metric Method for Minimization"; A.E.C. Research and Development Report, ANL-5990 (Rev.), 1959.

9. Davies, D., W. H. Swann; "Review of Constrained Optimization"; Optimization; Academic Press, London, 1969, pp. 187-202.

10. Dragomirescu, M.; "Theil-Van de Panne Algorithm for Convex Programming"; Studi si Cercetari Matematice, V19:5, 1967.

11. Esterby, B. E.; "Modification of Rosenbrock's Algorithm for the Nonlinear Programming Problem"; Masters Thesis, Georgia Institute of Technology, Atlanta, 1970.

12. Everett, H.; "Generalized Lagrange Multiplier Method for Solving Problems of Optimum Allocation of Resources"; Operations Research, V11:3, 399-418, 1963.

13. Fiacco, A. V., G. P. McCormick; "The Sequential Unconstrained Minimization Technique for Nonlinear Programming: A Primal-dual Method"; Management Science, V10, 360-366, 1964.

BIBLIOGRAPHY (Continued)

14. Fiacco, A. V., G. P. McCormick; "The Slacked Unconstrained Minimization Technique for Convex Programming"; SIAM Journal, V15:3, 505-515, 1967.

15. ______________________________; Nonlinear Programming Sequential Minimization Techniques; John Wiley and Sons, Inc., New York, 1968.

16. Fletcher, R.; "Function Minimization Without Evaluating Derivatives --A Review"; Computer Journal, V8, 1965.

17. ____________; "A Review of Methods for Unconstrained Optimization"; Optimization; Academic Press, London, 1969.

18. Fletcher, R., M. J. D. Powell; "A Rapidly Convergent Descent Method for Minimization"; Computer Journal, V6, 163, 1963.

19. Geoffrion, A. M.; "Reducing Concave Programs With Some Linear Constraints"; SIAM Journal of Applied Mathematics, V15:3, 653-664, 1967.

20. Glass, H., L. Cooper; "Sequential Search: A Method for Solving Constrained Optimization Problems"; Association for Computing Machinery Journal, V12, 71, 1965.

21. Goldfarb, D., L. Lapidus; "Conjugate Gradient Method for Nonlinear Programming Problems with Linear Constraints"; Industrial and Engineering Chemistry Fundamentals, V7, 1968.

22. Gue, R. L., M. E. Thomas; Mathematical Methods in Operations Research; The Macmillan Company, New York, 1968.

23. Hooke, R., T. A. Jeeves; "Direct Search Solution of Numerical and Statistical Problems"; Journal of the Association of Computing Machinery, V8, 212, 1961.

24. Kowalik, J., M. R. Osborne; Methods for Unconstrained Optimization Problems; American Elsevier Publishing Co., Inc., New York, 1968.

25. Lootsma, F. A.; "Constrained Optimization via Penalty Functions"; Philips Research Reports, V23, 408-423, 1968.

26. Lootsma, F. A.; "Constrained Optimization via Parameter-Free Penalty Functions"; Philips Research Reports, V23, 424-437, 1968.

27. Nelder, J. A., R. Mead; "A Simplex Method for Function Minimization"; Computer Journal, V7, 1965.

BIBLIOGRAPHY (Continued)

28. Pierre, D. A.; Optimization Theory with Applications; John Wiley and Sons, Inc., New York, 1969.

29. Powell, M. J. D.; "A Method for Nonlinear Constraints in Minimization Problems"; Conference on Optimization, Institute of Mathematics and Its Applications, 283-297, 1969.

30. ________________; "A Survey of Numerical Methods for Unconstrained Optimization"; Studies in Optimization 1; Society for Industrial and Applied Mathematics, Philadelphia, 1970.

31. Rosen, J. B.; "The Gradient Projection Method for Nonlinear Programming, Part I, Linear Constraints"; Journal of the Society for Industrial and Applied Mathematics, V8:1, 181, 1960.

32. Rosen, J. B., S. Suzuki; "Construction of Nonlinear Programming Test Problems"; Communications of the AMC, V8, 1965.

33. Rosenbrock, H. H.; "An Automatic Method for Finding the Greatest or Least Value of a Function"; The Computer Journal, V3:2, 175-184, 1960.

34. Sasson, A. M.; "Combined Use of the Powell and Fletcher-Powell NLP Methods for Optimal Load Flows"; IEEE Transactions on Power Apparatus and Systems, V88:10, 1530-1537, 1969.

35. Spendley, W., G. R. Hext, F. R. Himsworth; "Sequential Applications of Simplex Designs in Optimization and Evolutionary Operation"; Technometrics, V4, 1962.

36. Stocker, D. C., D. M. Himmelblau; "Applications of Optimization Techniques in Chemical Engineering, Part I"; American Institute of Chemical Engineers; 67th National Meeting, 1970.

37. Swenson, C. R.; Lecture Notes, School of Mathematics, Georgia Institute of Technology, Atlanta, 1971.

38. Swann, W. H.; "Report on the Development of a New Direct Search Method of Optimization"; I.C.I. Ltd. Central Instr. Lab. Res. Note 64/3, 1964.

39. Theil, H., C. Van de Panne; "Quadratic Programming as an Extension of Classical Quadratic Maximization"; Management Science, V7:1, 1-20, 1960.

BIBLIOGRAPHY (Concluded)

40. Van de Panne, C., A. Whinston; "The Symmetric Formulation of the Simplex Method for Quadratic Programming"; Econometrics, V37:3, 507-527, 1969.

41. Wolfe, P.; "The Simplex Method for Quadratic Programming"; Econometrics, V27, 382-398, 1959.

42. Wortman, J. D.; "NLPROG (a set of FORTRAN Programs to find the minimum of a Constrained Function)"; Ballistic Research Labs, Aberdeen Proving Ground, AD-684343, 1969.

43. Zangwill, W. I.; "Nonlinear Programming Via Penalty Functions"; Management Science, V13:5, 344-358, 1967.

44. ______________; Nonlinear Programming, A Unified Approach; Prentice-Hall, Inc., Englewood Cliffs, N. J., Chapter 12, 1969.

45. Zoutendijk, G.; Methods of Feasible Directions; Elsevier Publishing Co., Amsterdam, 1960.

46. ______________; "Nonlinear Programming: A Numerical Survey"; SIAM Journal of Control, V4, 1966.

47. ______________; "Computational Methods in Nonlinear Programming"; Studies in Optimization I; Society for Industrial and Applied Mathematics, Philadelphia, 1970.