Structural Optimization 8, 69–85 © Springer-Verlag 1994

Review Papers

Methods for optimization of nonlinear problems with discrete variables: a review

J.S. Arora and M.W. Huang

Optimal Design Laboratory, College of Engineering, The University of Iowa, Iowa City, IA 52242, USA

C.C. Hsieh

GM Systems Engineering, Troy, MI 48020, USA

Abstract The methods for discrete-integer-continuous variable nonlinear optimization are reviewed. They are classified into the following six categories: branch and bound, simulated annealing, sequential linearization, penalty functions, Lagrangian relaxation, and other methods. Basic ideas of each method are described and details of some of the algorithms are given. They are transcribed into a step-by-step format for easy implementation into a computer. Under "other methods", rounding-off, heuristic, cutting-plane, pure discrete, and genetic algorithms are described. For nonlinear problems, none of the methods are guaranteed to produce the global minimizer; however, "good practical" solutions can be obtained.

Notation

BBM	branch and bound method
$\mathbf{D}$	set of discrete values for all the discrete variables
D_i	set of discrete values for the i-th variable
d_{ij}	j-th discrete value for the i-th variable
f	cost function to be minimized
f^*	upper bound for the cost function
g_i	i-th constraint function
IP	integer programming
ILP	integer linear programming
L	Lagrangian
LP	linear programming
m	total number of constraints
MDLP	mixed-discrete linear programming
MDNLP	mixed-discrete nonlinear programming
n_d	number of discrete variables
NLP	nonlinear programming
p	number of equality constraints; acceptance probability used in simulated annealing
q_i	number of discrete values for the i-th variable
SLP	sequential linear programming
SQP	sequential quadratic programming
$\mathbf{x}$	design variable vector of dimension n
x_{iL}	smallest allowed value for the i-th variable
x_{iU}	largest allowed value for the i-th variable
∇	the gradient operator

1 Introduction

There are many cases in practical applications where the variables of optimization are not continuous. Some or all of the variables must be selected from a list of *integer* or *discrete* values. For example, structural members may have to be designed using sections available in standard sizes, member thicknesses may have to be selected from the commercially available ones, the number of bolts for a connection must be an integer, etc. Many of the existing optimization approaches focus only on problems with continuous design variables. Although this restricts their direct applicability to practical problems, they can be combined with other approaches to solve problems with discrete and integer variables.

Considerable interest was shown for discrete variable engineering optimization problems in the late 1960s and early 1970s. However, at that time optimization methods for continuous nonlinear programming (NLP) problems were not well developed, so the focus shifted to the development and evaluation of numerical algorithms for such problems. In the 1970s and 80s, substantial effort was put into developing and evaluating algorithms for continuous NLP problems. Although research in this area continues to develop better methods, especially for large scale problems, a few reliable algorithms are now available for general NLP problems. In this class are the sequential quadratic programming (SQP) and augmented Lagrangian methods (Arora 1989, 1990; Arora *et al.* 1991). Therefore, in recent years, the focus has shifted back to applications of optimization techniques to practical engineering problems that naturally use mixed discrete and continuous variables in their formulation.

The purpose of this paper is to define a general nonlinear optimization problem with mixed discrete and continuous design variables, and to review recent methods used to solve the problem. Only the methods relevant for nonlinear engineering applications are included in the review. Some of the approaches are described in detail and discussed herein. Since the variables that must have *integer* values can be considered as *discrete variables*, no distinction is made between the discrete and integer variables. Also *0-1 variable nonlinear optimization* (where a variable may or may not exist) is considered a special case of integer variable optimization with only two allowable values, 0 and 1.

Among the methods for discrete variable nonlinear optimization problems the following techniques have been most commonly discussed: branch and bound methods (BBM), zero-one variable techniques (integer programming), rounding-off techniques, cutting plane techniques, heuristic techniques, penalty function approach, Lagrangian relaxation (duality) approaches, and sequential linear programming (SLP). Each of them is not essentially a stand-alone method. Some of them can be combined to create an optimization algorithm for the mixed variable problems.

In Section 2 the discrete-continuous variable nonlinear optimization problem is defined. Section 3 contains an overview of the literature on the subject. In Sections 4 to 10 basic ideas of various methods for solving the mixed variable optimization problem are reviewed. Finally, Section 11 contains some concluding remarks.

2 Problem definition

Although the discrete variable problem appears to be easier to solve than the continuous one (since fewer possible solutions exist), in general, it is more difficult to solve except in some trivial cases. This is due to the fact that the discrete design space is disjoint and nonconvex. Thus, standard nonlinear programming techniques and optimality criteria cannot be applied directly.

An optimization problem can be expressed as a general *nonlinear programming (NLP) problem* of the following form:

$$\begin{aligned} &\text{minimize } f(\mathbf{x})\,,\\ &\text{subject to } g_i(\mathbf{x}) = 0\,, \quad i = 1,p\,,\\ &\qquad g_i(\mathbf{x}) \le 0\,, \quad i = p+1,m\,,\\ &\qquad x_{iL} \le x_i \le x_{iU}\,, \quad i = 1,n\,, \end{aligned} \tag{1}$$

where f and g are objective and constraint functions, respectively; x_{iL} and x_{iU} are lower and upper bounds for the variable x_i; p, m and n are the numbers of equality constraints, total constraints and design variables, respectively; and f and each g_i are usually assumed to be twice continuously differentiable.

When some of the variables are *discrete* and others are *continuous*, we have the so-called mixed-discrete nonlinear programming problem (MDNLP). The following formulation presents the mixed case of nonlinear programming problem (1):

$$\begin{aligned} &\text{minimize } f(\mathbf{x})\,,\\ &\text{subject to } g_i(\mathbf{x}) = 0\,, \quad i = 1,p\,,\\ &\qquad g_i(\mathbf{x}) \le 0\,, \quad i = p+1,m\,,\\ &\qquad x_i \in D_i \quad D_i(d_{i1}, d_{i2}, \dots, d_{iq_i})\,, \quad i = 1,n_d\,,\\ &\qquad x_{iL} \le x_i \le x_{iU}\,, \quad i = n_d+1,n\,, \end{aligned} \tag{2}$$

where n_d is the number of discrete variables, D_i is the set of discrete values for the i-th variable, q_i is the number of discrete values for the i-th variable, d_{ij} is the j-th discrete value for the i-th variable. In general, the number of available discrete values for each variable may be different.

It is important to understand some of the *features* of the MDNLP model (2). First, the inclusion of equality constraints in the problem may result in an *infeasible* problem, especially with only discrete variables. This is due to the reason that for the allowable discrete values, it may not be possible to satisfy the equality constraints. In such cases, a solution closest to the feasible domain may have to be accepted or the problem and the equality constraints have to be re-examined to provide a relaxed feasible domain. Second, any of the inequality constraints may not be active at the optimum point because the constraint surfaces may not pass through any of the discrete points. So, in numerical calculations only a point closest to the constraint boundary may be found. Third, there are no simple criteria to *terminate* the iterative search process. Thus, local optimality of the solution point cannot be assured unless an exhaustive search is performed. Fourth, the size of *discreteness* (i.e. how widely separated the allowable discrete values are) may govern the behaviour of some of the numerical algorithms as well as the final solution to the problem.

In many cases of design optimization, the designer must perform optimization over a range of alternative designs, say, several types of engines, connections, crank shafts, etc. Each of them has its own design variables and constraints, or it may have some physical interpretations for design variables and constraints as those of the other alternative designs, but they act as different ones when optimization is performed. This is sometimes referred to as topological decision-making (Sandgren 1990). Often the zero-one type variables are introduced to represent the choice among alternative designs (zero for not used; one for used). For example, if there are m alternatives, then m zero-one type variables are used. Only one of them will have nonzero values as the one to represent it being chosen as the design. Thus the sum of all zero-one type variables for this problem is always equal to one. Zero-one type variable techniques are usually implemented in conjunction with a branch and bound method.

Assuming that the design variables x_{ji} for all alternative designs are continuous variables (mixed variables may be included as explained above), the problem of choosing and optimizing a specific design can be formulated as follows:

minimize

$$f(\mathbf{x}) = z_1 f_1(\mathbf{x}_1) + z_2 f_2(\mathbf{x}_2) + z_3 f_3(\mathbf{x}_3) + \ldots + z_k f_k(\mathbf{x}_k)\,,$$

subject to

$$\begin{aligned} &z_j g_{ji}(\mathbf{x}_j) = 0\,, \quad i = 1,p_j\,, \quad j = 1,k\,,\\ &z_j g_{ji}(\mathbf{x}_j) \le 0\,, \quad i = p_j+1,m_j\,, \quad j = 1,k\,,\\ &\sum_{j=1}^{k} z_j = 1\,, \quad z_j = 0 \text{ or } 1\,, \quad j = 1,k\,,\\ &x_{jiL} \le x_{ji} \le x_{jiU}\,; \quad i = 1,n_j\,, \quad j = 1,k\,, \end{aligned} \tag{3}$$

where f_j is the objective function for the j-th alterative design $\mathbf{x}_k$, g_{ji} is the i-th constraint for the j-th alterative design, x_{ji} is the i-th design variable for j-th alterative design, x_{jiL} and x_{jiU} are lower and upper bounds for the variable x_{ji}, z_j is the zero-one type variable for the j-th alterative design, and p_j, m_j, and n_j are the numbers of equality constraints, total constraints and design variables for the j-th alterative design, respectively. An example illustrating this formulation can be found in the papers by Sandgren (1990a and b).

3 Overview of literature

A good overview of the methods for discrete variable optimization was recently presented by Bremicker *et al.* (1990), Loh and Papalambros (1991), and Vanderplaats and Thanedar (1991). The algorithms for mixed-discrete optimization problems are classified according to the type of variables that they can handle (Loh and Papalambros 1991). Binary (zero-one) versus non-binary, purely discrete versus mixed-discrete, and integer versus arbitrary discrete. Some methods cover more than one classification while others are strictly for a specific model. Loh and Papalambros (1991) first review the methods for mixed-discrete linear programming (MDLP) and then for mixed-discrete nonlinear programming (MDNLP). For MDLP, the methods reviewed are branch and bound, cutting plane, dynamic programming, implicit enumeration, and Lagrangian relaxation. A similar set of methods is then reviewed for MDNLP.

Branch and bound is perhaps the most widely known and used method for discrete optimization problems. It is basically an enumeration method where one first obtains a *minimizer* (minimum point) for the problem assuming all the variables to be continuous. Then each variable is assigned a discrete value in sequence and the problem is solved again in the remaining variables. This process of assigning discrete values to variables need not start from a continuous optimum point although that approach may reduce the number of times the problem needs to be re-solved to obtain a feasible discrete point and subsequently the optimum solution. It can be seen that the number of times the problem needs to be resolved increases exponentially with the number of variables. Several procedures have been devised to reduce this number (this is why it is sometimes called the implicit enumeration method); however, most of them do not guarantee a global solution to the nonlinear problem. The method was originally developed for LP problems; however, it is quite general and can be applied to nonlinear discrete and mixed variable problems.

Similar to the branch and bound method, other implicit enumeration techniques have been used by some researchers. Farkas and Szabo (1980) use a backtrack programming method to solve several structural optimization problems, such as a hybrid *I*-beam. Yuan *et al.* (1990) also use a backtrack programming method which adopts an interval halving strategy for efficiency of search. Bauer *et al.* (1981) and Pyrz (1990) apply a controlled enumeration method according to an increasing value of the objective function. Choi and Kwak (1990) use a direct search to find the discrete optimum solutions for reinforced concrete members by assuming that an optimum section exsists near the initially selected one. Mottl (1992) presents the voting method for a special class of non-linear programming problems.

Belegundu and Arora (1979) presented a survey of discrete variable optimization methods relative to the structural design problems. Plastic and elastic structural design problems were formulated and discussed. It turns out that plastic design of frames can be formulated as a linear programming (LP) problem. For discrete variable optimization, such problems can be transformed to integer (0-1) linear programming (ILP) problems. Many algorithms are available to solve such problems. The elastic structural design problem is nonlinear. For such problems, sequential linearization methods were discussed where the nonlinear problem was linearized and converted to an integer LP problem. The methods for nonlinear problems were classified into two broad categories: integer programming (IP), and non-integer programming such as branch and bound and heuristic methods. The IP techniques included the cutting plane method, branch and bound (implicit enumeration), group theoretic techniques, network approach, and heuristic (approximate) techniques.

Vanderplaats and Thanedar (1991) classify the discrete variable structural optimization methods into three categories: branch and bound, approximation, and *ad hoc* methods. In the approximation methods, the branch and bound method is used on an approximate problem after a continuous solution has been obtained for the original nonlinear problem. The approximate problem can be defined in several ways. For example, as a linearized problem using Taylor's expansion followed by solution using discrete linear programming methods, or conservative approximations which result in nonlinear approximate problems having explicit variables. *Ad hoc* methods consist of several different strategies. For example, May and Balling (1992) propose to use simulated annealing on the approximate problem after a continuous solution has been obtained. The allowable set of discrete values for each variable is reduced to only three values in the neighbourhood of a continuous solution. The penalty functions approach is also classified as an *ad hoc* approach. Shin *et al.* (1990) introduce a sine penalty function to penalize the nondiscrete values for the variables. Fu *et al.* (1991) have also used the penalty function approach to solve five small scale engineering optimization problems. The difficulties with this approach are the introduction of additional local minima and repeated minimizations by adjusting the penalty parameters. The dual strategy proposed by Schmit and Fleury (1980), which has its roots in the Lagrangian relaxation concept (Geoffrion 1974), is also classified as an *ad hoc* approach. In this approach, a separable approximation to the original nonlinear problem is first created and is then solved by the dual approach. The dual problem can be solved in a closed form. The method is not good if the original problem is not well approximated as a separable problem, or the discrete values for the variables are widely separated.

It is observed that some of the methods for discrete variable optimization use the structure of the problem to speedup the search for the discrete solution. These methods are not suitable for implementation into a general purpose application software. The branch and bound method and simulated annealing are the most general methods; however, they are also the most time-consuming. In the present paper, the numerical methods for solving MDNLP are classified as follows:

1. branch and bound methods,
2. simulated annealing,
3. sequential linearization method – followed by branch and bound, integer programming, simulated annealing, and others,
4. penalty function approach,
5. Lagrangian relaxation methods, and
6. other methods – rounding-off, cutting plane, heuristics, genetic algorithms, pure discrete techniques, etc.

These methods are described in some detail in the subsequent sections.

4 Branch and bound method (BBM)

4.1 The basic method

The first use of the branch and bound method is attributed to Land and Doig (1960) for linear problems. Dakin (1965) later modified the algorithm which has been subsequently used by many researchers. Garfinkel and Nemhauser (1972) have also described the use of BBM on convex linear programming problems. Reinschmidt (1971) used BBM to solve the integer linear programming problem related to the plastic design of frames. The problem was posed as a linear programming problem and solved by a modified version of Geoffrion's (1967, 1969) implicit enumeration method for discrete programming. This algorithm, designed to solve linear 0-1 programming problems, is based on an algorithm due to Balas (1965). If q is the number of discrete values for each variable, then Geoffrion's method will have a maximum of 2^{qn} combinations. None of the algorithms ever enumerates all the possible combinations. Reinschmidt (1971) also applied the algorithm for nonlinear problems where the sequential linearization procedure was used.

The following definitions are useful for the description of the branch and bound (implicit enumeration) method (Reinschmidt 1971).

1. *Half-bandwidth.* It is useful to limit the number of discrete values for each variable. This can be done by starting with a good initial solution, such as a point obtained by rounding-off the continuous solution, and limiting the range of allowable discrete values. For example, one could take r values below and $r - 1$ values above the current value, giving a range of $2r$ discrete values. The parameter r is called the half-bandwidth.
2. *Completion.* The assignment of discrete values from the allowable ones to all the variables.
3. *Feasible completion.* A completion that satisfies all the constraints.
4. *Partial solution.* The assignment of discrete values to some but not all the variables; at least one variable is unassigned, or free.
5. *Fathoming.* A partial solution is said to be fathomed if it is determined that no matter how assignments of discrete values to the remaining free variables are made, it is impossible to find a feasible completion of smaller cost function value than the one previously known. If a partial solution is fathomed, it implies that all possible completions of this partial solution have been *implicitly enumerated*, and therefore need not be explicitly enumerated. A key to the efficiency of the procedure is a set of strong rules which will fathom the partial solutions.

Gupta and Ravindran (1983) use Dakin's method together with a generalized reduced gradient method, to solve nonlinear mixed-integer problems. Mesquita and Kamat (1987) combine branch and bound with an SQP method to solve the problem of stiffened laminated composite plates. Sandgren (1990) used an approach similar to that by Gupta and Ravindran, but included equality constraints having zero-one type variables for topological optimization problems. Branch and bound was combined with the exterior penalty function and sequential quadratic programming methods to treat the mixed-discrete NLP problem. John *et al.* (1988) combine BBM with sequential linearization for the discrete optimal design of trusses. Hajela and Shih (1990) use BBM to solve multiobjective optimization problems with discrete and integer variables. Recently, Salajegheh and Vanderplaats (1993) have used BBM for optimizing truss structures with discrete sizing and shape variables. Large storage space and an exponential growth in computational effort limits the applicability of BBM to higher dimensional problems.

The usual BBM approach is to systematically search continuous solutions where the discrete variables are forced to take discrete values from the specified set. The logical structure for the set of solutions is that of a tree for each variable. Initially an optimum point is obtained by treating all the variables as continuous. If this solution is discrete, then the process is terminated. If one of the desired variables is not discrete, then its value lies between two discrete values, e.g. $d_{ij} < x_i < d_{ij+1}$. Now two subproblems are defined, one with the constraint $x_i \le d_{ij}$ and the other with $x_i \ge d_{ij+1}$. This process is called *branching*. It basically eliminates some portion of the continuous feasible region which is not feasible for the discrete problem. It does not, however, eliminate any of the discrete feasible solutions. The two subproblems are solved again, and the optimum solutions are stored as nodes of the tree containing optimum values of the variables, the cost function and the appropriate bounds on the variables.

This process of branching and solving continuous problems described in the foregoing is continued until a feasible solution to the MDNLP is obtained. The cost function corresponding to this solution becomes an *upper bound* on the optimum solution for the original MDNLP. From this point onwards, all the nodes of the tree that have cost function values higher than the established upper bound are eliminated from further consideration. Such nodes are said to have *fathomed*; i.e. reached their lowest point and no further branching is necessary from them. The process of fathoming the nodes is called *bounding*.

The process of branching and fathoming is repeated from each of the unfathomed nodes. From each node, at most two new nodes may originate. A better upper bound for the optimum cost function is established when a *feasible discrete solution* is obtained with the cost function smaller than the current upper bound. The nodes may be fathomed in any of the following three ways: (i) feasible discrete solution to the continuous solution with the cost function value higher than the current upper bound, (ii) infeasible continuous problem, and (iii) the optimal value of the cost function for the continuous problem higher than the upper bound. The search for the optimum terminates when all the nodes have fathomed.

It is important to note that the BBM is guaranteed to find the global optimum only if the problem is linear or convex. In the case of general nonlinear nonconvex problems, it is possible that a node is fathomed too early and one of its branches actually contains the true global solution.

As an illustrative example, consider the following integer LP problem (Belegundu and Arora 1979):

minimize $f = -20x_1 - 10x_2$,

subject to $g_1 = -20x_1 - 10x_2 + 75 \le 0$,

$$g_2 = 12x_1 + 7x_2 - 55 \leq 0\,,$$
$$g_3 = 25x_1 + 10x_2 - 90 \leq 0\,,$$
$$x_1,\ x_2 \geq 0 \text{ and integers}.$$

Figure 1 shows how a continuous solution (a node or a candidate) is branched and how new constraints are added into the subproblems. Subproblem 4 shows how a node is terminated.

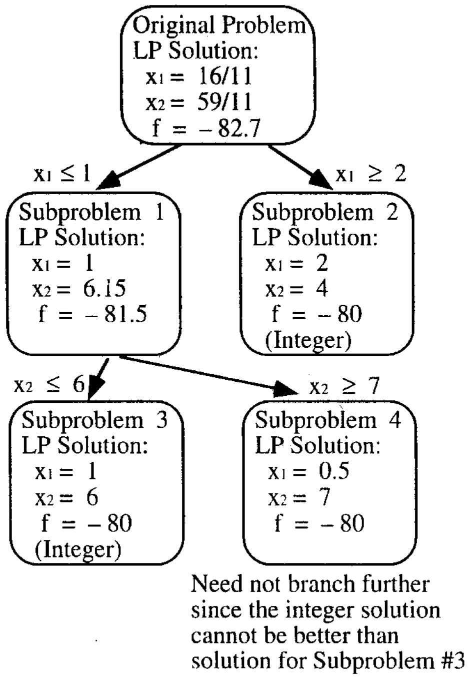

Fig. 1. Example of branch and bound technique

There are many variations of BBM. Two basic strategies, "depth first search" and "breadth first search" were discussed by Ringertz (1988), each having its advantages and drawbacks. The following two algorithms are generalized from the work of Ringertz (1988) and Sandgren (1990a, b), respectively.

Algorithm A: **BBM** [based on the paper by Ringertz (1988)]

Step 1. Assume that all design variables are continuous and solve problem (1).

Step 2. If the solution is discrete, stop; otherwise, continue.

Step 3. Calculate an upper bound f^* for the cost function which is obtained by rounding off the continuous solution (obtained in Step 1) or by a heuristic method (both will be discussed in the following sections).

Step 4. Branch on the variable closest to its discrete value and solve the two subproblems.

Step 5. Add candidates (nodes) to the list that stores nodes ready to be selected.

Step 6. If list is empty, stop; otherwise, continue.

Step 7. Select the last candidate in the list.

Step 8. Define the constraint set S_j for the last candidate which is obtained by imposing the correct simple bounds.

Step 9. Solve problem (1) for the last candidate $\mathbf{x}$ with $\mathbf{x} \in S_j$.

Step 10. If the solution is feasible for the discrete problem, continue; otherwise, go to Step 6 to proceed to the next candidate.

Step 11. If the solution is feasible and discrete, the upper bound f^* for the cost function value f is updated $f^* = \min(f, f^*)$, where f is the cost function value obtained in Step 9. Go to Step 6. If the solution is feasible but not discrete, continue.

Step 12. It is investigated if further branching can result in a better discrete solution. If $f > f^* + \Delta f$ (the cost function value f is larger than the upper bound by a certain value Δf, indicating no need for further branching on this subproblem), go to Step 6. Otherwise, continue.

Step 13. Define candidates and add them to the list. As in Step 4, branch on the variable closest to being discrete. Two candidates are obtained separating on the variable closest to being discrete. Go to Step 6.

Algorithm B: **BBM** [based on the papers by Sandgren (1990a, b)]

Step 1. Assume all the design variables to be continuous and solve (1).

Step 2. Determine the type of design variable. If the variable is required to be an integer, then proceed to Step 3. If the variable is required to be discrete, then proceed to Step 4. If the variable is continuous, then proceed to Step 5.

Step 3. The design variable which is required to be an integer is used for branching in the following manner. Let the value of the design variable be separated into two parts, P and Q, where P is the integer part and Q is the fractional part. If P is defined as the largest integer not exceeding $P+Q$, then the region $P < x_i < P+1$ contains no feasible integer values, and two new branches can be created by imposing the restrictions $x_i \leq P$ on one branch and $x_i \geq P+1$ on the other.

Step 4. The design variable that is required to have a discrete value is used for branching in the following manner. Let the value of the design variable be represented by the value R. If the discrete value d_k is defined as the largest discrete value not exceeding R, and d_{k+1} is defined as the smallest discrete value exceeding R, the region $d_k < R < d_{k+1}$ contains no feasible discrete value. Thus, two new branches are created by imposing the restrictions $x_i \leq d_k$ on one branch and $x_i \geq d_{k+1}$ on the other.

Step 5. If an integer or discrete solution is generated, the result becomes an upper bound on the final value of the objective function. All nodes (candidates) with a value grater then this upper bound may be eliminated from the search (assuming the minimization problem and unimodality).

Step 6. If all nodes have been eliminated, the process is terminated. If a node still exists, then the next required integer or discrete variable is used for branching. Then Steps 2, 3, and 4 are repeated until all required integer or discrete variables have been branched upon, with the upper bound on the objective function always being updated to the best integer or discrete variable solution found.

Step 7. Once all the required integer or discrete variables have been branched upon and a branching node still exists, additional branching is initiated. This step is repeated until all nodes are eliminated from further branching.

4.2 Some enhancements of the method

The branch and bound method has been used successfully to deal with discrete design variable problems and has proved to be quite robust. However, for problems with large numbers of discrete design variables, the number of subproblems (nodes) becomes large, making the method inefficient. This drawback is more severe for nonlinear optimization problems. Several enhancements of the method are possible. For example, Mesquita and Kamat (1987) suggest fixing the variable that is used for branching to its proper value for the two subproblems, and eliminating it from further consideration. This reduces dimensionality of the problem which can result in efficiency. As the iterative process progresses, more and more variables are fixed and size of the optimization problem keeps on reducing.

Recently, Tseng *et al.* (1992) carried out an extensive study to modify BBM to improve its efficiency. Two aspects have been studied: the order of branching the variables and the branching algorithm. They are summarized in the sequel.

4.2.1 Order of branching of the variable

Referring to Step 11 of Algorithm A, an upper bound for the cost function is established once a feasible discrete solution is obtained (as in Subproblem 3 of Fig. 1). This upper bound is then used to eliminate other branches which have a larger cost function value or have the same cost function value but whose solutions are not discrete (as in Subproblem 4 of Fig. 1). One can eliminate more branches if a smaller upper bound is generated as early as possible. This can be accomplished by choosing the right variable for branching at each step. Many algorithms with different branching orders for variables have been proposed by researchers. Tseng *et al.* (1992) have proposed the following 5 criteria and determined from their numerical experience that criterion No. 5 is the most suitable.

1. *The min-clearance method.* Calculate the clearances between the non-discrete optimum values for the variables and their nearest allowable values. Choose a variable with the minimum clearance for the next branching step. The criterion is expressed as follows:

$$\Delta x_i = \min(|x_i - d_{ij}|, |x_i - d_{ij+1}|),$$

$$x_{\text{branch}} = \min(\Delta x_1, \Delta x_2, \ldots, \Delta x_n), \tag{4}$$

where x_i, d_{ij}, d_{ij+1}, and Δx_i are the optimum value, the lower allowable value, the upper allowable, and the clearance for the i-th design variable, respectively; x_{branch} is the variable for further branching.

2. *The max-clearance method.* Calculate the clearances between the non-discrete optimum values for the variables and their nearest allowable values. Choose a variable with the maximum clearance for the next branching step. The criterion is expressed as follows:

$$\Delta x_i = \max(|x_i - d_{ij}|, |x_i - d_{ij+1}|),$$

$$x_{\text{branch}} = \max(\Delta x_1, \Delta x_2, \ldots, \Delta x_n). \tag{5}$$

3. *The min-clearance difference method.* Calculate the clearances between the non-discrete optimum values for the variables and their nearest allowable values. Choose a variable with the minimum clearance difference for the next branching step. The criterion is defined as follows:

$$\Delta x_i = \big||x_i - d_{ij}| - |x_i - d_{ij+1}|\big|,$$

$$x_{\text{branch}} = \min(\Delta x_1, \Delta x_2, \ldots, \Delta x_n). \tag{6}$$

4. *The max-clearance difference method.* Calculate the clearances between the non-discrete optimum values for the variables and their nearest allowable values. Choose a variable with the maximum clearance difference for the next branching step. The criterion is defined as follows:

$$\Delta x_i = \big||x_i - d_{ij}| - |x_i - d_{ij+1}|\big|,$$

$$x_{\text{branch}} = \max(\Delta x_1, \Delta x_2, \ldots, \Delta x_n). \tag{7}$$

5. *The max-cost difference method.* This method evaluates the cost function at new nodes directly by assigning each variable the lower or upper bound value. The criterion is defined as

$$\text{Cost}_{i\ell} = f(x_1, x_2, \ldots, d_{ij}, \ldots, x_n),$$

$$\text{Cost}_{iu} = f(x_1, x_2, \ldots, d_{ij+1}, \ldots, x_n),$$

$$\Delta\text{Cost}_i = |\text{Cost}_{i\ell} - \text{Cost}_{iu}|,$$

$$x_{\text{branch}} = \max(\Delta\text{Cost}_1, \Delta\text{Cost}_2, \ldots, \Delta\text{Cost}_n), \tag{8}$$

where $\text{Cost}_{i\ell}$ and Cost_{iu} are the estimated cost function values of the first and second nodes for the i-th design variable, respectively; ΔCost_i is the estimated cost difference for the i-th design variable when it is assigned to its lower or upper discrete value.

4.2.2 Branching algorithm

Tseng *et al.* (1992) also tested three different algorithms for branching a node. The idea of branch algorithms is to avoid the middle nodes (subproblems) of the branched trees. However, their numerical experience showed that the last two algorithms are not as good as expected and hence only the single branch algorithm is suggested.

1. *Single branch algorithm.* In each branch procedure, only one discrete design variable is treated, and a new constraint is added to the new subproblems (as shown in Fig. 1).
2. *Multi-branch algorithm.* In each branch procedure, all discrete design variables are treated and hence it generates 2^n subproblems, where n is the number of discrete design variables.
3. *Unbalanced branch algorithm.* A combination of the first two algorithms. The first algorithm is used to generate two branches by treating only one discrete variable. One of the two branches is treated as a subproblem and by using the second algorithm, another 2^{n-1} subproblems are generated.

5 Simulated annealing

Simulated annealing is a simple technique that can be used to find a global minimizer for continuous-discrete-integer nonlinear programming problems. The approach does not require continuity of differentiability of the problem functions because it does not use any gradient or Hessian information.

The basic idea of the method is to generate random points and evaluate the problem functions. If the trial point is infeasible, it is rejected right away. If the trial point is feasible and the cost function value is smaller than its current best value, then the point is accepted, and the best cost funtion value is updated. If the point is feasible but the cost function value is higher than the best value known so far, then the question is whether to accept or reject the point. The answer is that it is sometimes accepted and sometimes rejected. The acceptance is based on the value of the probability density funtion of the Bolzman-Gibbs distribution. If this probability density function has a value (this is called *acceptance probability*) greater than a random number, then the trial point is accepted as the best solution even if its cost function value is higher than the known best value. In computing the probability density function, a parameter called the *temperature* is used. For the optimization problem, this temperature can be the *target value for the cost function* corresponding to the global minimizer. Initially, a larger target value is selected. As the trials progress, this target value is reduced (this is called the *cooling schedule*), and the process is terminated after a fairly large number of trials. The acceptance probability steadily decreases to zero as the temperature is reduced. Thus in the initial stages, the method is likely to accept worse designs, while in the final stages the worse designs are almost always rejected. This strategy avoids getting trapped at local minimizers. The main deficiencies of the method are the unknown rate at which the target level is to be reduced, and the uncertainty in the total number of trials and in the number of trials after which the target level needs to be reduced.

The idea of simulated annealing originated in statistical mechanics (Metropolis *et al.* 1953). The approach is ideally suited for solving combinatorial optimization problems, such as the discrete-integer programming problems. However, first attempts in the use of simulated annealing for solving hard combinatorial optimization problems (the travelling salesman problem) were made in the 1980s by Kirkpatrick *et al.* (1983) and Cerny (1985). The problem of combinatorial optimization can be related to find a configuration *at the ground state of a matter* that occurs at very low temperatures. The ground state corresponds to a state with the absolute lowest internal energy. In a practical context, low temperature is not a sufficient condition for finding a ground state of the matter. Experiments that determine the low temperature state of a material are done by careful *annealing*; for example, growing a crystal from a metal. First the material is melted, then the temperature is lowered slowly to cool the metal. This process is repeated several times. Longer time is spent at temperatures in the vicinity of the freezing point than at higher temperatures. If this is not done, i.e. time is not spent long enough at each temperature so that the equilibrium is not reached, the resulting crystal may have many defects or the material may even become glass with no crystalline order. This corresponds to metastable state. Corresponding configurations will correspond to only locally optimal structures.

This annealing can be simulated on a computer using the procedure described before to find a ground state of a material. In this way we can compute configurations and other properties at the ground state of a material using statistical mechanics. This is termed *simulated annealing*. The sequence of temperatures and the number of rearrangements of the configurations attempted to reach equilibrium at each temperature is considered as an *annealing schedule.*

The idea of combinatorial optimization and its similarity with simulated annealing in statistical mechanics becomes clear from the classical travelling salesman problem: given a list of N cities and a means of calculating the cost of travelling between any two cities, one must plan the salesman's route which passes through each city once and returns finally to the starting point while minimizing the total cost. All exact methods known for determining an optimal route require a computing effort that increases exponentially with N. Iterative improvement, commonly applied to such problems, is much like the microscopic rearrangement process modelled by statistical mechanics. However, accepting only rearrangements that lower the cost function of the system is like extremely rapid quenching from high temperatures to very low temperatures. Thus what we obtain is a system corresponding to a metastable state, i.e. a local minimum. Optimization procedures that borrow this simulated annealing idea from statistical mechanics, such as in Metropolis' algorithm, provide a generalization of iterative improvement in which *controlled uphill steps* can also be incorporated in the search for a better solution.

Here we give an example of a simulated annealing algorithm (based on the work by Kincaid and Padula 1990).

Algorithm C: simulated annealing

Step 1. Choose initial temperature T_0 (expected global minimum for the cost function) and a feasible trial point $\mathbf{x}^{(0)}$. Compute $f[\mathbf{x}^{(0)}]$. Select ILIM (a larger integer representing a limit on the number of trials to reach the expected minimum value), $0 < \text{TFACT} < 1$, and TLIMIT (a lower limit for the cost function used as the stopping criterion); TFACT is used to reduce the expected minimum value. Initialize the iteration counters as $j = 0$ and $i = 1$.

Step 2. Generate randomly a new point $\mathbf{x}^{(i)}$. For problems having discrete design variables, each discrete value for a specific discrete design variable has equal opportunity to be selected for constructing the random point. If the point is infeasible, generate another random point until feasibility is satisfied. Generate a random number z uniformly distributed in $(0, 1)$. Compute $f[\mathbf{x}^{(i)}]$ and $\Delta f = f[\mathbf{x}^{(i)}] - f[\mathbf{x}^{(0)}]$.

Step 3. If $\Delta f < 0$, then take $\mathbf{x}^{(i)}$ as the new trial configuration $\mathbf{x}^{(0)}$, set $f[\mathbf{x}^{(0)}] = f[\mathbf{x}^{(i)}]$ and go to Step 4. Otherwise, calculate

$$p(\Delta f) = \exp(-\Delta f / T_j)\,. \tag{9}$$

If $z < p(\Delta f)$, then take $\mathbf{x}^{(i)}$ as the new trial configuration $\mathbf{x}^{(0)}$ and go to Step 4. Otherwise go to Step 2.

Step 4. If $i <$ILIM, then set $i = i + 1$ and go to Step 2. If $i >$ ILIM and $T_j <$TLIMIT, then stop. Otherwise, go to Step 5.

Step 5. Set $T_{j+1} = T_j *$TFACT; set $j = j + 1$, $i = 1$ and go to Step 2.

For general optimization problems, temperature is an arbitrary parameter that is taken in the same units as the cost function. It depends on the mean values of the cost function. This parameter is sometimes called effective temperature. The energy corresponds to the cost function. The configuration has quite a different meaning for different optimization problems.

There are continuous global optimization methods that are also based on ideas in statistical mechanics (Aluffi-Pentini *et al.* 1985); for example, simulated annealing may be combined with a local minimization procedure. However, in that case the temperature T is continuously decreased very slowly so that the effect is similar to annealing. Using the probability density function given in (9), a criterion has been developed to decide whether or not to start a local search from a particular point $\mathbf{x}$ (Lucidi and Piccioni 1989). Recently, May and Balling (1992) proposed a filtered simulated annealing strategy. Based on known gradient information, this method can probabilistically filter out many of the poorer candidates.

6 Integer programming

Optimization problems where the variables are required to take on integer values are called integer programming (IP) problems. If some of the variables are continuous, then we have mixed variable problems. If all the functions of the problem are linear, we have an integer linear programming (ILP) problem, otherwise it is nonlinear. Nonlinear problems can be solved by sequential linearization procedures. Therefore, we concentrate on linear problems only.

The ILP problem can be converted to 0-1 programming problem. Several algorithms are available to solve such problems. Therefore, we first discuss transformation of ILP to a 0-1 programming problem. Consider an LP defined as follows:

$$\text{minimize} \quad f = \mathbf{c}^T\mathbf{x}\,,$$

$$\text{subject to } \mathbf{AX} \le \mathbf{b}$$

$$x_i \ge 0,\ \text{integer}\,, \quad i = 1, n_d\,,$$

$$x_{iL} \le x_i \le x_{iU}\,, \quad i = n_d + 1, n\,. \tag{10}$$

Define z_{ij} as the 0-1 variables. Then the i-th integer variable can be expressed as

$$x_i = \sum_{j=1}^{q_i} z_{ij} d_{ij}\,. \tag{11}$$

Substituting this into the foregoing mixed ILP problem we have

$$\text{minimize} \quad f = \sum_{i=1}^{n_d} c_i \left[\sum_{j=1}^{q_i} z_{ij} d_{ij}\right] + \sum_{k=n_d+1}^{n} c_k x_k\,,$$

$$\text{subject to} \quad \sum_{j=1}^{n_d} a_{ij} \left[\sum_{m=1}^{q_j} z_{jm} d_{jm}\right] + \sum_{k=n_d+1}^{n} a_{ik} x_k \le b_i\,,$$

$$\sum_{j=1}^{q_i} z_{ij} = 1,\ i = 1, n_d,\ z_{ij} = 0 \text{ or } 1 \text{ for all } i \text{ and } j\,,$$

$$x_{iL} \le x_i \le x_{iU}\,, \quad i = n_d + 1, n\,. \tag{12}$$

The branch and bound methods can be used to solve the 0-1 ILP problem. Other algorithms are (Gue *et al.* 1968) as follows.

1. Balas Additive Algorithm (1965). This is basically a branch and bound algorithm. A unique "value criterion" is used to decide if a particular variable should take on the value of 0 or 1.
2. Glover Algorithm (1965). The concept of a "*surrogate constraint*" is defined and the corresponding surrogate problem is solved.
3. Balas Filter Algorithm (1967). In this approach, the earlier additive algorithm is modified to incorporate the surrogate constraint concept.
4. Lawler-Bell Algorithm (1966). An algorithm is developed for solving general zero-one programming problems. The cost and constraint functions are not required to be linear functions of the zero-one variables.
5. Geoffrion Algorithm (1967, 1969). Implicit enumeration algorithms are presented based on Balas additive algorithm. The surrogate constraint concept is also used. Most branch and bound algorithms for the zero-one programming problem can be described by the general framework proposed by Geoffrion (1967).

It is important to note that many modern computer programs for linear programming have an option to solve discrete variable LP problems; e.g. LINDO (Schrage 1983).

7 Sequential linearization methods

A popular approach with many researchers to solve the MDNLP has been to linearize the nonlinear problem at the current estimate of the minimizer. Then a discrete-integer linear programming method is used to solve the linearized subproblem. There are several ways in which the linearized subproblems can be defined and solved. In this section, we review some of the procedures that have been developed and evaluated.

Many researchers transform the linearized subproblem to a binary variable (0-1) problem (Templeman and Yates 1983; Ming-Zhu 1986). The number of variables is increased considerably; however, many methods have been developed to solve integer linear programming problems. Many LP codes have an option of solving such problems, so the MDNLP can be solved using the sequential LP approach and existing codes. A modification of this approach is to obtain a continuous optimum point first, and then linearize and use integer programming methods (Olsen and Vanderplaats 1989). This process can reduce the number of integer LP problems to be solved. The size of the LP problem can also be reduced by restricting the number of discrete values to be in the neighbourhood of the continuous solution.

Ghattas and Grossman (1991) discussed discrete variable structural optimization problems. They formulated the problem as the so-called simultaneous analysis and design problem. In this formulation, the design variables and the state variables (nodal displacements, stresses) are treated simultaneously as variables of optimization. All the state equations are treated as equality constraints. The formulation results in a large scale NLP problem; however, it is sparse. It does not require any special procedures for design sensitivity analysis; i.e. gradients of all the constraints and the cost function with respect to all the variables can be calculated quite readily. The discrete variable problem thus formulated is then converted to 0-1 programming problem (binary LP) and solved using an integer LP algorithm. The approach is demonstrated on two example problems of truss design. In the allowable set of discrete values for each variable, zero value is also included. This allows some of the members of the structure to be eliminated, so topology of the structure can also be optimized; i.e. topological optimization problems can be solved with the simultaneous formulation. Such problems cannot be solved easily by the so-called nested approach where only the design variables are treated as variables of optimization. The reason is that with nested formulation, singularities occur when certain members attain zero values. The branch and bound method was also used to solve the problem and compare results. The proposed method was more efficient.

Balling and co-workers (Hager and Balling 1988; May and Balling 1991, 1992) have discussed the discrete variable optimization problem relative to the design of steel frames. They first obtain a continuous optimum point using an NLP algorithm (the SQP method). Then approximation concepts are used to convert the problem to an explicit design problem. In one approach, the problem is converted to a linear program at the continuous solution. The LP problem is then solved using the branch and bound approach with the number of discrete values for each variable restricted to the nearest three or four values. After a discrete solution has been obtained, a new linear approximation is constructed and the process is repeated one more time. In another approach, the simulated annealing (Kincaid and Padula 1990; Balling 1991) was used after a base design (perhaps a continuous solution) had been obtained and an explicit approximate problem had been generated. The simulated annealing requires repeated testing of random trial designs. The trial designs were restricted to a neighbourhood of the base design. The approximate analysis approach was used to evaluate the functions in the neighbourhood of the base design. A smart simulated annealing approach was also presented to sieve out some undesirable designs, speeding-up the iterative process.

Bremicker *et al.* (1990) also used sequential linearization followed by BBM to solve discrete variable structural optimization problems. The linearized discrete problem is solved by the simplex method to obtain information at each node of the tree. Their approach is compared with the pure BBM, where the SQP method is used to solve the nonlinear problem to obtain information at each node. The study shows the SLP method to be superior to the pure BBM. In another study, Loh and Papalambros (1991) proposed a different procedure for solving MDNLP. In this procedure, the problem is first linearized in terms of the discrete variables only keeping the continuous variables fixed at their current values. The linearized subproblem is solved using the BBM and the simplex method. Then the discrete variables are kept fixed at their current values, and the continuous subproblem is solved using the GRG method, although other methods may also be used. Good results are reported with this procedure.

The existence of discrete variables does not fundamentally change the SLP procedure. Once the LP has been constructed, a reliable method such as branch and bound or zero-one binary variable approach can be used to solve the problem. An SLP algorithm together with BBM to solve nonlinear mixed-discrete structural optimization problems is given in the following.

Algorithm D: SLP and BBM [based on the paper by Bremicker *et al.* (1990)]

Step 1. Choose a starting point $\mathbf{x}^{(0)}$, and let the iteration counter $k = 0$.

Step 2. Given an iteration point $\mathbf{x}^{(k)}$, compute f, g, ∇f, ∇g.

Step 3. Generate a linear subproblem L_k as follows:

$$\text{minimize } \nabla f[\mathbf{x}^{(0)}]^T[\mathbf{x}-\mathbf{x}^{(0)}],$$

$$\text{subject to } g_j[\mathbf{x}^{(0)}]+\nabla g_j[\mathbf{x}^{(0)}]^T[\mathbf{x}-\mathbf{x}^{(0)}] \le 0, \quad (13)$$

where $\mathbf{x} \in \mathbf{D}$, the discrete set. Like the continuous SLP method, the move limits must be adjusted to achieve convergence (Arora 1989). Thus the following constraints are added:

$$\boldsymbol{\ell}^{(k)} \le \mathbf{x}^{(0)} \le \mathbf{u}^{(k)},$$

with

$$\ell_{ik} = \max\{x_{ik} + \frac{\alpha}{2}(u_i - \ell_i), \ell_i\}, \quad i = 1, n,$$

$$u_{ik} = \min\{x_{ik} + \frac{\alpha}{2}(u_i - \ell_i), u_i\}, \quad i = 1, n. \quad (14)$$

One way to choose α is by the following recursive formula:

$$\text{if } k = 0, \ \alpha = 0,$$

$$\text{otherwise, } \alpha_{\text{new}} = \alpha_{\text{old}}/(1+\alpha_{\text{old}}). \quad (15)$$

Step 4. Solve L_k by the branch and bound and simplex methods.

Step 5. Check the following stopping criterion, i.e. if $\| \mathbf{x}^{(k)} - \mathbf{x}_{\text{best}} \| < \delta$, then stop; otherwise, continue. Here δ is a small value, and $\mathbf{x}_{\text{best}}$ denotes the ε-feasible point with the lowest objective function value that has been found in previous iterations. A point $\mathbf{x}^{(k)}$ is said to be ε-feasible for the MDNLP model if it satisfies the discreteness requirement and

$$\sum_1^m \Delta g_j \le \varepsilon, \quad \Delta g_j = g_j, \quad g_j > 0. \quad (16)$$

Step 6. Select a new point $\mathbf{x}^{(k+1)}$, let $k = k+1$ and go on with Step 2. A new point $\mathbf{x}^{(k)}$ is accepted as the new $\mathbf{x}_{\text{best}}$ when either $f_k \le f_{\text{best}}$, if $\mathbf{x}^{(k)}$ is ε-feasible, or

$$\Big\{\sum_1^m \Delta g_j\Big\}_k \le \Big\{\sum_1^m \Delta g_j\Big\}_{k-1},$$

$$\text{if } \mathbf{x}^{(k-1)} \text{ is } \varepsilon-\text{feasible}. \quad (17)$$

In Step 5, the convergence criteria implies that two different subproblems yield approximately the same solutions and the current solution can be treated as the optimal solution. Also note that the usual optimality conditions for the continuous NLP cannot be used in the case of MDNLP.

The above algorithm will usually converge faster if the problem is pure discrete. For the case of MDNLP, the authors suggested the following modification for Step 4.

Step 4a. Solve L_k by the branch and bound and simplex methods for discrete variables.

Step 4b. Solve the following subproblem for continuous variables $\mathbf{x}_c$ with fixed discrete variables $\mathbf{x}_d$:

$$\underset{\mathbf{x}_c}{\text{minimize}} \quad f(\mathbf{x}_d, \mathbf{x}_c)\,,$$
$$\text{subject to} \quad \mathbf{g}(\mathbf{x}_d, \mathbf{x}_c) \le \mathbf{0}\,. \tag{18}$$

The SLP algorithm where LP is solved using a 0-1 integer variable approach is given below.

Algorithm E: SLP + zero-one variables [based on the paper by Olsen and Vanderplaats (1989)]
The linearized form of problem (2) is given as

$$\text{minimize} \;\; f(\mathbf{x}) \approx f[\mathbf{x}^{(0)}] + \nabla f[\mathbf{x}^{(0)}]^T[\mathbf{x} - \mathbf{x}^{(0)}]\,,$$
$$\text{subject to } g_i(\mathbf{x}) \approx g_i[\mathbf{x}^{(0)}] + \nabla g_i[\mathbf{x}^{(0)}]^T[\mathbf{x} - \mathbf{x}^{(0)}]\,,$$
$$x_i \in D_i\,, \quad i = 1, n_d\,, \quad x_{iL} \le x_i \le x_{iU}\,, \quad i = n_d + 1, n. \tag{19}$$

Substituting zero-one type variables into problem (19), the problem becomes

minimize

$$f(\mathbf{x}) \approx f[\mathbf{x}^{(0)}] + \sum_{i=1}^{n_d} \frac{\partial f}{\partial x_i}\Big[\Big(\sum_{j=1}^{q_i} z_{ij} d_{ij}\Big) - x_i^{(0)}\Big] + \sum_{k=n_d+1}^{n} \frac{\partial f}{\partial x_k}[x_k - x_k^{(0)}]\,,$$

subject to

$$g_j(\mathbf{x}) \approx g_j[\mathbf{x}^{(0)}] + \sum_{i=1}^{n_d} \frac{\partial g_i}{\partial x_i}\Big[\Big(\sum_{m=1}^{q_i} z_{im} d_{im}\Big) - x_i^{(0)}\Big] + \sum_{k=n_d+1}^{n} \frac{\partial g_i}{\partial x_k}[x_k - x_k^{(0)}]\,, \quad \sum_{j=1}^{q_i} z_{ij} = 1\,, \quad i = 1, n_d\,, \; z_{ij} = 0$$
$$\text{or } 1 \;\text{ for all } i \text{ and } j\,, \quad x_{iL} \le x_i \le x_{iU}\,, \quad i = n_d + 1, n. \tag{20}$$

With the above formulation, the problem is now in a form suitable for mixed-integer LP techniques. Two problems arise, the first one is how to obtain the initial value of $\mathbf{x}^{(0)}$. We assume that the discrete optimum is in the vicinity of the continuous optimum obtained by treating all design variables as continuous ones. Therefore $\mathbf{x}^{(0)}$ is taken as the continuous optimum. However, this approach may not yield a feasible discrete solution. The other approach to overcome this difficulty is to round-off the continuous optimum into a feasible discrete solution, and thus obtain $\mathbf{x}^{(0)}$. The second difficulty is that the problem will become large if some design variables have a large number of discrete variables. Olson and Vanderplaats (1989) suggest the use of a truncated set of discrete variables. A truncated set has the following justifications.

1. It is assumed that the present solution $\mathbf{x}^{(0)}$ is in the vicinity of the discrete optimum. Discrete values distant from the values in $\mathbf{x}^{(0)}$ are unlikely members of the optimum set.
2. The Taylor series approximation is most valid in the vicinity of the present $\mathbf{x}^{(0)}$. Move limits of some sort should be imposed to prevent extreme constraint violation.
3. Inclusion of more possible discrete values in the approximate problem does not necessarily improve the final solution to the true problem.
4. The truncated set reduces the size of the approximate problem and hence improves efficiency.

8 Penalty function approach

Another approach to treat the requirement of discreteness is to define additional constraints and construct a penalty function for them. This term imposes penalty for deviations from the discrete values. Davydov and Sigal (1972) applied this concept on convex problems with uniformly spaced discrete intervals. Gisvold and Moe (1972) utilized a similar approach for discrete variables with arbitrarily spaced intervals. Recently, Shin *et al.* (1990) applied the penalty function approach to the optimization of truss structures. Fu *et al.* (1991) used a similar approach for more general problems. Additional penalty terms are added in the original penalty function to reflect the requirement that design variables have discrete values. This approach is described by a step by step algorithm in the sequel.

The general NLP problem defined in (1) without discrete variables can be replaced by an unconstrained minimization problem as follows:

$$\Phi(x, r) = f(x) + r \sum_{j=1}^{m} y(g_j)\,, \tag{21}$$

where r is the penalty multiplier for the constraints, $y(g_j)$ can be chosen as $[\max\{-g_j(\mathbf{x}), 0\}]^2$ for an exterior penalty, $1/g_j(\mathbf{x})$ for an interior penalty, or a linear or quadratic extension function for an extended interior penalty technique. For the discrete variable problem defined in (2), a modified augmented functional to include discrete variable information is expressed as follows (Shin *et al.* 1990):

$$\Psi(\mathbf{x}, r, s) = f(\mathbf{x}) + r \sum_{j=1}^{m} y(g_j) + s \sum_{i=1}^{n_d} z_i(\mathbf{x})\,, \tag{22}$$

where s is the penalty multiplier for non-discrete values of the design variables, m and n_d are the number of constraints and number of discrete design variables, respectively, and $z_i(\mathbf{x})$ is the penalty term for non-discrete values of the i-th design variables. Shin *et al.* (1990) used the following form for $z_i(\mathbf{x})$:

$$z_i(\mathbf{x}) = \frac{1}{2}\left\{\sin\left(\frac{2\pi\{x_i - \frac{1}{4}(d_{ij+1} + 3d_{ij})\}}{d_{ij+1} - d_{ij}}\right) + 1\right\}, \tag{23}$$

where $d_{ij} < x_i < d_{ij+1}$. This function only penalized non-discrete design variables and assures continuity of the first derivatives of the function in (22) at the discrete values of the design variables. A criterion,

$$\varepsilon_c = \frac{\text{amount of constraint penalty}}{\text{cost function}}\,, \tag{24}$$

is used to denote the tolerance which shows how close the design approaches to the continuous optimum before activating the discrete penalty.

Algorithm F: penalty function method [based on the paper by Shin *et al.* (1990)]

Step 1. Form $\Phi(\mathbf{x}, r)$ for regular penalty function approach.
Step 2. Minimize $\Phi(\mathbf{x}, r)$.

Step 3. If criterion (24) is satisfied, continue. Otherwise, decrease the penalty multiplier r and go to Step 1.

Step 4. Freeze r and initialize s. Form $\Psi(\mathbf{x}, r, s)$.

Step 5. Minimize $\Psi(\mathbf{x}, r, s)$.

Step 6. If the solution is feasible, continue. Otherwise, increase r, re-initialize s and go to Step 5.

Step 7. If the design is sufficiently close to prescribed discrete value, stop. Otherwise, increase the penalty multiplier s and go to Step 5.

9 Lagrangian relaxation approach

The Lagrangian relaxation method is similar to the penalty function method. The main difference is that the additional terms due to discrete variables are added to a Lagrangian function instead of a penalty function. The method was introduced by Geoffrion (1974) for linear integer programming problems. The basic idea of the method can be tracked back to the surrogate concept of Glover (1968). In this approach, the constraints are scaled, added to the cost function and removed (relaxing) from the problem, giving an unconstrained problem. The optimal cast function for the relaxed problem gives a lower bound to the optimum cost function for the original problem. However, to obtain a better bound, the dual of the relaxed problem must be solved. This dual is discontinuous and requires methods for nonsmooth optimization. Lagrangian relaxation is best suited in conjunction with branch and bound. It is not suitable with the SLP method due to poor estimates of the Lagrange multipliers.

Schmit and Fleury (1980) used this approach to solve nonlinear discrete structural optimization problems. The original problem is replaced by a sequence of convex and separable approximate subproblems. The Lagrangian duality is used at each iteration to solve the subproblem in which the dual function is maximized using a subgradient method. The subproblem is convex in the continuous case, but is no longer convex in the discrete case. The dual solution may correspond to a large set of possible solutions, and the true discrete optimum may not be a member of this set. An explicit enumeration is used to obtain an approximate solution. Sepulveda and Cassis (1986) presented a similar approach but used a possibly more efficient algorithm to maximize the dual function. Ringertz (1988) used a Lagrangian relaxation method to solve six structural optimization problems. He also stated several types of Lagrangian functions. Grierson and co-workers (Cameron *et al.* 1991; Grierson and Lee 1984; Grierson and Cameron 1989) also used this dual approach to solve optimal design problem of 2D and 3D steel frames. Recently, a similar approach was used by Jonsson and Larsson (1990) to solve the discrete sizing problems and by Bauer (1992) to optimize a circular plate with stepwise varying thickness.

Several methods have been developed by using generalized or augmented Lagrangian approaches for continuous nonconvex NLP problems. One type of generalized Lagrangian is (Ringertz 1988):

$$L(\mathbf{x}, \lambda, \mu) = f(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i g_i + \sum_{i=1}^{m} \mu_i P[g_i^+(\mathbf{x})]\,, \tag{25}$$

where $g_i^+(\mathbf{x}) = \max\{g_i(\mathbf{x}), 0\}$, λ_i are the Lagrange multipliers, $\mu_i > 0$ are the multipliers and P is a continuous function satisfying:

(i) $P(u) \geq 0$ for all u,

(ii) $P(u) = 0$ if and only if $u = 0$. (26)

Examples of such functions are

$$P(u) = u^2\,, \quad P(u) = |u|\,, \quad P(u) = \cosh(u) - 1\,,$$
$$P(u) = \exp(u^2) - 1\,. \tag{27}$$

Schmit and Fleury (1980) used the ordinary Lagrangian function

$$L(\mathbf{x}, \lambda) = f(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i g_i\,, \tag{28}$$

and solved the corresponding dual problem

maximize $\Phi(\lambda)$, $\lambda_i \geq 0$, $i = 1, m$,

where

$$\Phi(\lambda) = \underset{\mathbf{x} \in \mathbf{D}}{\text{minimize}}\, L(\mathbf{x}, \lambda)\,. \tag{29}$$

Ringertz (1988) used the following Lagrangian function:

$$L(\mathbf{x}, \mu) = f(\mathbf{x}) + \sum_{i=1}^{m} \mu_i (\cosh[g_i^+(\mathbf{x})] - 1)\,, \tag{30}$$

for which the dual problem becomes

maximize $\Phi(\mu)$,

where

$$\Phi(\mu) = \underset{\mathbf{x} \in \mathbf{D}}{\text{minimize}}\, L(\mathbf{x}, \mu)\,. \tag{31}$$

The subgradient of $\Phi(\mu)$ is

$$\partial_i \Phi = \cosh[g_i^+(\mathbf{x})] - 1\,, \tag{32}$$

which is already available when $\Phi(\mu)$ is evaluated. The above dual problem is concave, continuous and piecewise linear in the purely discrete case. A subgradient method can be used to solve this dual problem.

Algorithm G: subgradient method [based on the book by Minoux (1986)]

Step 1. Give an initial estimate for the dual variables $\mu^{(0)}$. Set $k = 0$.

Step 2. Compute $\Phi[\mu^{(k)}]$ and $\partial\Phi$ using (31) and (32). If $\partial\Phi = 0$ (corresponding to a feasible solution), stop. Otherwise, continue.

Step 3. Take a step in the subgradient direction:

$$\mu^{(k+1)} = \mu^{(k)} + \alpha_k \partial\Phi\,. \tag{33}$$

A reasonable choice of step size α_k can be provided as follows:

$$\alpha_k = \{\Phi^* - \Phi[\mu^{(k)}]\} / \parallel \partial\Phi^{(k)} \parallel^2\,, \tag{34}$$

where Φ^* is an estimate of the dual maximum and can be taken as the cost function value obtained by rounding off the continuous optimal solution.

Step 4. $k = k + 1$, and go to Step 2.

One difficulty of the above algorithm is the unconstrained primal minimization over the discrete set $\mathbf{D}$ in (31). To overcome such problems, various algorithms have been proposed, such as the heuristic approaches discussed in the next section, and the neighbourhood search approach. A neighbourhood search algorithm can be found in the paper by Reiter and Sherman (1965).

The Lagrangian relaxation method does not guarantee to find the global optimum since the subproblems are nonconvex when the discrete variables are introduced.

10 Other approaches

There are many other *ad hoc* methods that have been tried for discrete-integer programming problems. For example, direct search methods have also been suggested, for example, by Reiter and Rice (1966), and Glankwahmdee *et al.* (1979) have suggested discrete gradient methods. Simplex type methods have been described by Fox and Liebmann (1981) who have successfully solved several test problems. An adaptive random discrete search was implemented by Kelahan and Gaddy (1978) and demonstrated on four MDNLP problems. An adaptive search technique has also been suggested by Arora (1989). In this approach, the continuous optimum is used as a starting point. The variable that is closest to its discrete value is fixed and the problem is re-optimized treating other variables as continuous. The process is continued until all the discrete variables have been fixed. Reasonable performance has been reported with the procedure. Some of these techniques are described in this section.

10.1 Rounding-off technique

A simple approach for discrete variable problems is to obtain an optimum solution using the continuous approach, and then use rounding-off and heuristics to obtain a discrete optimum solution. The variables are rounded-off to their nearest available discrete values once the solution is found. Although this idea is simple, it always results in an infeasible design for problems having large numbers of variables. However, it is not necessary to increase all variables into their upper discrete neighbours. Some of them could be decreased to their lower neighbour. The main problem of the rounding-off approach turns out to be the selection of variables to be increased or decreased. Ringertz (1988) provided a brief discussion for making "good" rounding decisions. Other rules can be found in the book by Siddal (1982, pp. 223-266). Glover and Sommer (1975), and Beveridge and Schechter (1970) noted that this strategy may have difficulty in converging, especially in the case of high nonlinearity and widely separated allowable discrete values. The discrete minimizer need not be in the neighbourhood of a continuous optimizer.

A possible implementation of the rounding-off approach can be expressed as follows.

Algorithm H: rounding-off method [based on the paper by Ringertz (1988)]

```
Step 1. Set $n$ = number of design variables.
Step 2. Assume all the design variables to be continuous and
        solve problem (2).
Step 3. If the solution is discrete, stop. Otherwise, continue.
Step 4. FOR $k = 1, n$
        Calculate the perturbation of the Lagrange function
        $\Delta L = L[\mathbf{x}^{(k)}, \lambda] - L(\mathbf{x}, \lambda) =$
        $\{f[\mathbf{x}^{(k)}] + \sum_{i=1}^{m} \lambda_i g_i[\mathbf{x}^{(k)}]\} - \{f(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i g_i(\mathbf{x})\}$, (35)
        where $\lambda$ and $\mathbf{x}$ are the Lagrange multipliers and de-
        sign variables obtained in Step 2, respectively, and
        $\mathbf{x}^{(k)}$ are the design variables obtained in Step 2
        with the $k$-th variable perturbed to its upper discrete
        neighbour.
        END FOR
Step 5. Choose the design variables $\mathbf{x}^{(k)}$ which minimize
        $L[\mathbf{x}^{(k)}, \lambda]$, as the current design and discard the $k$-th
        variable from being a design variable. Set $n = n - 1$
        and if $n = 1$, stop, otherwise, go to Step 2.
```

An alternative to (35), according to Luenberger (1984, pp. 338-341), is as follows:

$$\Delta L = \frac{1}{2} \Delta x_k^2 \frac{\partial^2 L(\mathbf{x}, \lambda)}{\partial x_k^2} . \tag{36}$$

The number of continuous problems which needs to be solved by the above algorithm is $(n - 1)$. However, the number of design variables is reduced by one for each iteration.

10.2 Cutting-plane technique

This approach was first proposed by Gomory (1958, 1963) for integer solutions to linear programs. Kelley (1960) developed the idea further for convex problems and called it the cutting-plane technique. Other variations have been presented (Salkin 1975). The basic idea is to solve the noninteger linear programming problem and to derive and add constraints to the problem to force out the integer solution. At each iteration, an additional linear constraint is added that discards (cuts) a portion of the continuous feasible region. Toakley (1971) solved the linear programming problem of the optimal design of statically determinate trusses and plastic design of frames by applying the cutting plane technique together with a branch and bound algorithm.

The idea of the cutting plane technique can be explained by using a two-variable LP problem with x_1 as a continuous variable and x_2 restricted to be an integer variable (Toakley 1971). First the LP problem is solved by treating x_2 as a continuous variable. Referring to Fig. 2, the solution will occur at one of the vertices of the polygon ABCDEF. However, since x_2 is an integer variable, the actual solution will be on the lines cd, be, fh or ag. Usual LP methods such as the simplex method cannot be used since the foregoing line segments do not form a convex set. However, by considering the shaded area enclosed by a convex boundary, it can be seen that the optimal solution is at one of the vertices of the polygon abcdefg.

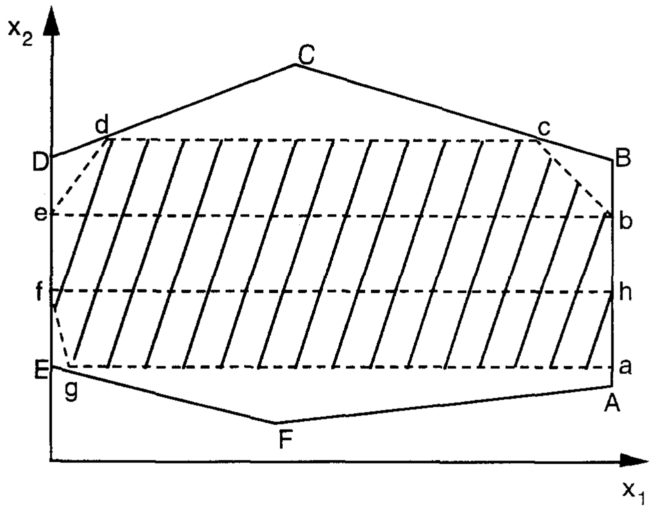

Fig. 2. Cutting-plane technique with one integer variable

Gomory's algorithm generates the additional constraints or cutting planes required to produce integer solutions. The algorithm first obtains a continuous optimal solution by the simplex method. If the integer variables do not have integer values, a new constraint is added and the dual simplex algorithm (Hadley 1962) is then used to re-optimize the simplex tableau. This process of adding constraints and re-optimizing is continued until the feasible optimal solution is obtained.

There is little interest in the cutting plane approach for practical problems, even for the linear ones. The reason is that a large number of cuts is needed to force out the discrete solution, even for moderate size problems. Another problem is that the cuts tend to be parallel or nearly parallel causing numerical difficulties.

10.3 Heuristic techniques

The use of BBM becomes computationally impractical when the number of discrete design variables is large. The heuristic rules which help in making decisions about rounding-off the continuous solution to the next feasible discrete point are then used to overcome this drawback of BBM. Cella and Logcher (1971) have stated a heuristic algorithm and applied it to the elastic frame and truss problem. Linearized structural equations are used to accelerate reanalysis. The algorithm uses branch and bound ideas along with Balas' (1965) filter method of 0-1 programming. Some heuristic rules can also be found in the paper by Ringertz (1988).

Lin and Kernighan (1973) proposed a heuristic approach which has been extended and applied to mechanism problems by Lee and Freudenstein (1976). Hua (1983) used heuristics to reduce the number of combinations in the explicit enumeration process. The algorithm is suitable for structural problems only since characteristics of stress and displacement constraints are used in developing heuristics.

Recently, Cha and Mayne (1989) combined heuristics with the reliable sequential quadratic programming method for mixed discrete optimization problems. Their algorithm does not require a continuous optimum point to initiate the search for the discrete solution. A feasible discrete solution is, however, required to initiate the procedure. Therefore, a separate algorithm is used to locate a feasible discrete starting point. Once a feasible point is found, the subsequent improved points are forced to remain feasible and discrete. Therefore, the algorithm is based on an interior search strategy. The ideas of regional and neighbourhood searches are used in the algorithm to locate the possible optimal point. In the regional search, large steps are possible. It is proposed to use the sequential quadratic programming (SQP) algorithm with rank-one update for the Hessian of the Lagrangian, although any other algorithm for continuous NLP problems can be used. A search direction is computed by solving the QP subproblem. Then a discrete line search is performed using several heuristics to see if an improved discrete point can be found. If such a point is found then the SQP strategy is continued; otherwise, the algorithm switches to the neighbourhood search strategy. During this search, projected gradient direction and discrete line search procedures are used to locate better discrete points. Many heuristics are used at this stage; e.g. second-order approximation of the cost function, search along the coordinate directions, search in the discrete neighbourhood, etc. Rank-one updating for the Hessian of the cost function is also maintained along with the Hessian of the Lagrangian that is used in the SQP method. Several heuristic stopping criteria are suggested to terminate the search. As for most other algorithms, the current algorithm is not guaranteed to find the true local minimum point. It has been evaluated on 25 test problems and has worked very well. It is more efficient than some other algorithms.

10.4 Genetic algorithms

These methods are in the category of stochastic search methods, such as the simulated annealing (Goldberg and Kuo 1987; Hajela 1989), in that both methods have their basis in *natural processes.* Genetic algorithms are also implicit enumeration procedures. Their philosophical basis is in Darwin's theory of the survival of the fittest. A set of design alternatives representing a population in a given generation are allowed to *reproduce* and *cross* among themselves, with bias allocated to the most fit members of the population. A combination of the most desirable characteristics of mating members of the population results in progenies that are more fit than their parents. Thus, if a measure which indicates the fitness of a generation is also the desired goal of a design process, successive generations produce better values of the cost function. An advantage of this approach is that no gradient information is needed. Therefore, differentiability requirements – needed in gradient based methods – can be relaxed.

In a genetic algorithm, one starts with a set of designs. From this set, new and better designs are reproduced using the fittest members of the set. Each design must be represented by a finite length string. Usually binary strings have been used for this purpose. The entire process is similar to a natural population of biological creatures, where successive generations are conceived, born and raised until they are ready to reproduce. Three operators are needed to implement the algorithm: (1) reproduction, (2) crossover, and (3) mutation.

Reproduction is an operator where an old string is copied into the new population according to the string's fitness. Here, fitness is defined according to the cost function value. More fit strings (those with smaller cost function values) receive higher numbers of offsprings. There are many different strategies to implement this reproduction operator.

The next operator – *crossover* – corresponds to allowing selected members of the population to exchange characteristics of the design among themselves. Crossover entails selecting a start and end position on a pair of mating strings at random, and simply exchanging the string of 0's and 1's (for a binary string) between these positions on one string with that from the mating string. This is akin to transfer of genetic material in biological reproduction processes facilitated by DNA strings.

Mutation is the third step in this genetic refinement process, and is one that safeguards the process from a complete premature loss of valuable genetic material during reproduction and crossover. In terms of a binary string, this step corresponds to selecting few members of the population, determining at random a location on the strings, and switching the 0 or 1 at that location.

The foregoing three steps are repeated for successive generations of the population until no further improvement in the fitness is attainable. The member in this generation with the highest level of fitness is the optimum design. A possible implementation of the rounding-off approach can be expressed as follows.

Algorithm I: genetic algorithm [based on the work by Lin and Hajela (1992) and Sugimoto (1992)]

Step 1. Convert the constrained problem to an unconstrained problem using the exterior penalty function formulation:

minimize $F^* = F + P$,

where F is the cost function for the original constrained problem, and P is the penalty term based on the constraints.

Step 2. Define a schema, an L-digit genetic string with 0 or 1 for each digit to represent a design point, where L is the total number of digits required for a design point. A genetic string is constructed by linking n binary strings together, where n is the number of design variables and a binary string is a 0-1 string which represents a design variable. An m-digit binary string has 2^m possible 0-1 combinations that represent the discrete values of a discrete design variable. The following method can be used to convert a 0-1 combination to its corresponding discrete value of a discrete design variable with n_d discrete values.

Calculate m which is the smallest integer satisfying $2^m > n_d$.

Calculate the integer j:

$$j = \sum_{i=1}^{m} \mathrm{ICHAR}(i) 2^{(i-1)} + 1 . \tag{37}$$

If $j > n_d$,

$$j = \mathrm{INT}\Big(\frac{n_d}{2^m - n_d}\Big)(j - n_d) , \tag{38}$$

where ICHAR(i) is the value of the i-th digit (either 0 or 1) of the 0-1 combination, and INT $(\frac{n_d}{2^m - n_d})$ is the integer part of $\frac{n_d}{2^m - n_d}$.

Thus the j-th discrete value corresponds to this 0-1 combination.

Step 3. Randomly generate N_p genetic strings (members of the population) according to the schema, where N_p is the population size.

Set $K = 0$, and $I_c = 0$.

Step 4. Define the fitness function as follows:

$$f_i = F^*_{\max} - F^* , \tag{39}$$

where f_i is the fitness of the i-th design, and $F^*_{\max}$ is the maximum value of F^*.

Step 5. Assign fitness values of all genetic strings.

Set $K = K + 1$.

Step 6. Reproduction: select the more fit members of the population into the mating pool, from which members are selected for crossover and mutation transformations. Each member in the original population has a probability of selection $f_i / \sum_{j=1}^{N_p} f_j$. A selected member is copied to the mating pool (a new population with the same size, N_p) and still remains in the original population for further selection. Thus the new population may contain identical members and may not contain members found in the original population. The average fitness will be increased, however.

Step 7. Crossover: select two mating parents from the mating pool. Randomly choose two sites on the genetic strings and swap strings of 0's and 1's between two chosen sites among the mating pair. A probability of crossover p_c is defined to determine if crossover should be implemented. Set $I_c = I_c + 1$.

Step 8. Mutation: choose a few members from the mating pool according to the probability of mutation p_m, and switch a 0 to 1 or vice versa at a randomly selected site on each chosen string.

Step 9. If $I_c < I_{\max}$, where $I_{\max}$ is an integer defined by the user, go to Step 7. Otherwise, continue.

Step 10. If any of the convergence criteria are satisfied, stop. Otherwise, go to Step 5.

The convergence criteria can be defined as follows (Sugimoto 1992).

1. The number of the generations K reaches 50.
2. The number of the strings having highest fitness exceeds 10 percent of the population size.
3. The best value of the fitness is not updated in the consecutive 20 generations.

It is important to note that the genetic algorithm is not a random walk or random search method. The latter method generally reduces to an exhaustive enumeration of the design space. In contrast, the genetic algorithm uses random choice as a tool to manipulate and direct the search in a region which is most likely to contain the global optimum.

10.5 Pure discrete technique

In this approach, all variables of the problem are required to have discrete values. Therefore, even the continuous variables are transformed to have values at a fixed discrete increment. Certain care must be exercised in selecting the increment to maintain continuous nature of the variables in analysis. With all variables being discrete, a pure discrete optimization algorithm is then used to solve the problem. Bauer *et al.* (1981) use a discrete method to solve lattice structure optimization problems. They transform the mixed nonlinear programming problems into discrete nonlinear programming problems. Amir and Hasegawa (1989) also present a discrete variable optimization algorithm based on discrete gradients and apply it to structural optimization problems.

11 Discussions and conclusions

For convex problems, the branch and bound method has a solid theoretical basis and fathoming rule which guarantees global minimum. For non-convex problems, it may end up fathoming nodes that should not have been fathomed. However, a useful solution can usually be obtained for such cases. As mentioned previously, the most obvious disadvantage of BBM is the computational effort when the number of variables is large. Practically, BBM is most suitable when the number of discrete design variables is small, say less than 30 or so.

The zero-one type variable technique is suitable for problems with alternative design selections; e.g. choosing from several types of engines. However, the problem will become large when the number of selections is large since each selection (alternative design) introduces its own set of design variables. Also, a zero-one type variable is not appropriate for nonlinear terms in cost or constraint functions.

The main advantage of the rounding-off approach is the number of subproblems to be solved which grows linearly with the number of design variables. However, with a large number of design variables, the selection of the variable for rounding-off becomes more difficult as some design constraints always tend to be violated.

Simulated annealing and genetic algorithms have some advantages: they do not require gradients of functions and convexity of the problem, they do not need a subproblem solver such as for an NLP algorithm, and they find global minima. However, they are the least efficient methods since the cost and constraint function values have to be calculated for a large number of times. They can be reasonably effective if the function evaluation time is not large and search domain can be reduced.

The Lagrangian relaxation approach does not guarantee finding a global solution, even if it is a convex problem before discrete variables are introduced. For the penalty function approach, the most important aspect is the treatment of the penalty parameters s and r. For example, if s is too large at the first few discrete iterations, the design can be trapped at a local minimum or, if r is increased, the design may move away from the optimum. The parameters λ and μ in the Lagrangian relaxation appproach have a similar effect.

An efficient way to solve discrete design variable problems is to transform the nonlinear programming problems (NLP) into sequential linear programming problems (SLP) or sequential quadratic programming problems (SQP). Existing powerful software for continuous optimizations of SLP and SQP problems can be modified according to the previous discussion and implemented on problems having discrete variables. This may be preferable for future studies. When dealing with certain well-behaved problems, the numerical optimality criteria methods may be more efficient to solve the subproblem in SLP and SQP methods. A tool to deal with discrete variables is still necessary, such as the robust old-fashioned BBM, the inexpensive approximate methods such as rounding-off and heuristic techniques, or the penalty function method and the Lagrangian relaxation method, which are usually efficient and reliable if considerable user insight and good parameter adjustments are used.

In conclusion, it appears that the branch and bound, linearization, and some heuristic methods hold promise for general purpose engineering applications. Some of the methods may be combined to improve efficiency of the solution process. In the branch and bound methods, a key to efficiency is to establish good upper bounds to the discrete solution as quickly as possible, and select appropriate discrete variables for branching. In this respect, the recent work of Tseng *et al.* (1992) can be very useful. In addition, the first-order information available on the continuous minimizer can be useful. These aspects are topics for further investigations.

Acknowledgements

The authors would like to thank Professors R. Balling, A. Belegundu, O. Ghattas and P. Papalambros, and Dr. P. Thanedar for reading a draft of the paper and providing comments to improve the material. We are also thankful to the reviewers for providing useful comments.

References

Allufi-Pentini, F.; Parisi, V.; Zirilli, F. 1985: A global optimization and stochastic differential equations. *J. Optim. Theory and Appl.* **47**, 1-16

Amir, H.M.; Hesagawa, T. 1989: Nonlinear mixed-discrete structural optimization. *J. Struct. Engng., ASCE* **115**, 626-646

Arora, J.S. 1989: *Introduction to optimal design.* New York: McGraw–Hill

Arora, J.S. 1990: Computational design optimization: a review and future directions. *Struct. Safety* **7**, 131-148

Balas, E. 1965: An additive algorithm for solving linear problems with zero-one variables. *Operations Research* **13**, 1485-1525

Balas, E. 1991: Discrete programming by the filter method. *Operations Research* **5**, 915-958

Balling, R.J. 1991: Optimal steel frame design by simulated annealing. *J. Struct. Eng.* **117**, 1780-1795

Bauer, J. 1992: Algorithms of nondifferentiable optimization in discrete optimum structural design. *ZAMM* **72**, 563-566

Bauer, J.; Gutkowski, W.; Iwanow, Z. 1981: A discrete method for lattice structures optimization. *Eng. Opt.* **5**, 121-128

Belegundu, A.D.; Arora, J.S. 1979: Discrete variable optimization in structural engineering: a survey. *Technical Report No. 49*, Materials Engineering, The University of Iowa

Beveridge, G.S.; Schechter, R.S. 1970: *Optimization: theory and practice.* New York: McGraw-Hill

Bremicker, M.; Papalambros, P.Y.; Loh, H.T. 1990: Solution of mixed-discrete structural optimization problems with a new sequential linearization algorithm. *Comp. & Struct.* **37**, 451-461

Cameron, G.E.; Xu, L.; Grierson, D.E. 1991: Discrete optimal design of 3d frameworks. In: Ural, O.; Wang, T.-L. (eds.) *Electronic computation*, pp. 181-188. New York: ASCE

Cella A.; Logcher, R. 1971: Automated optimum design from discrete components. *J. Struct. Div., ASCE* **97**, 175-188

Cerny, V. 1985: Thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm. *J. Optim. Theory Appl.* **45**, 41-51

Cha, J.Z.; Mayne, R.W. 1989: Optimization with discrete variables via recursive quadratic programming. *J. Mech. Des. ASME* **111**, 124-136

Cha, J.Z.; Mayne, R.W. 1991: The symmetric rank one formula and its application in discrete nonlinear optimization. *J. Mech. Des. ASME* **113**, 312-317

Choi, C.K.; Kwak, H.G. 1990: Optimum RC member design with predetermined discrete sections. *J. Struct. Engng.* **116**, 2634-2655

Dakin, R.J. 1965: A tree-search algorithm for mixed integer programming problems. *Comput. J.* **8**, 250-255

Davydov, E.G.; Sigal, I. 1972: Application of penalty function method in integer programming problems. *Engng. Cybernetics* **10**, 21-24

Farkas, J.; Szabo, L. 1980: Optimum design of beams and frames of welded I-sections by means of backtrack programming. *Acta Tech. Acad. Sci. Hung.* **91**, 121-135

Fleury, C.; Braibant, V. 1986: Structural optimization – a new dual method using mixed variables. *Int. J. Num. Meth. Engng.* **23**, 409-428

Fox, D.B.; Liebmann, J.S. 1981: A discrete nonlinear simplex method for optimized engineering design. *Engng. Optim.* **5**, 129-149

Fu, J.-F.; Fenton, R.G.; Cleghorn, W.L. 1991: A mixed integer-discrete-continuous programming method and its application to engineering design optimization. *Engng. Optim.* **17**, 263-280

Garfinkel, R.; Nemhauser, G. 1972: *Integer programming.* New York: John Wiley & Sons

Geoffrion, A.M. 1967: Integer programming by implicit enumeration approach for integer programming. *Operations Research* **9**, 178-190

Geoffrion, A.M. 1969: An improved implicit enumeration approach for integer programming. *Operations Research* **17**, 437-454

Geoffrion, A.M. 1972: Generalized Bender's decomposition. *J. Optim. Theory Appl.* **2**, 82-114

Geoffrion, A.M. 1974: Lagrangian relaxation for integer programming. *Mathematical Programming Study* **10**, 237-260

Geoffrion, A.M.; Marsten, R.E. 1972: Integer programming algorithms: a framework and state-of-the-art survey. *Management Science* **18**, 465-491

Ghattas, O.N.; Grossman, I.E. 1991: MINLP and MILP strategies for discrete sizing structural optimization problems. In: Ural, O.; Wang, T.-L. (eds.) *Electronic computation*, pp. 181-188. New York: ASCE

Gisvold, K.M.; Moe, J. 1972: A method for nonlinear mixed integer programming and its application to design problems. *J. Engng. Ind. ASME* **94**, 353-364

Glankwahmdee, A.; Liebmann, J.S.; Hogg, G.L. 1979: Unconstrained discrete nonlinear programming. *Eng. Opt.* **4**, 95-107

Glover, F. 1965: A multiphase-dual algorithm for the zero-one integer programming problem. *Operations Research* **13**, 879-919

Glover, F. 1968: Surrogate constraints. *Operations Research* **16**, 741-749

Glover, F.; Sommer, D. 1975: Pitfalls of rounding in discrete management decision problems. *Decision Science* **22**, 43-50

Goldberg, D.E.; Kuo, C.H. 1987: Genetic algorithms in pipeline optimization. *J. Computing in Civil Engng.* **1**, 128-141

Gomory, R.E. 1958: An algorithm for integer solutions to linear programs. *Technical Report 1,* Princeton-IB; Mathematical Research Project

Gomory, R.E. 1963: An algorithm for integer solutions to linear programs. In: Graves, G.W.; Wolfe, P. (eds.) *Recent advances in mathematical programming.* New York: McGraw-Hill

Grierson, D.E.; Cameron, G.E. 1989: Microcomputer-based optimization of steel structures in professional practice. *Microcomputers in Civil Engineering* 4

Grierson, D.E.; Lee, W.H. 1984: Optimal synthesis of steel frameworks using standard sections. *J. Struct. Mech.* **12**, 335-370

Gue, R.L.; Liggett, J.C.; Cain, K.C. 1968: Analysis of algorithms for the zero-one programming problem. *Comm. Assoc. Computing Machinery* **11**, 837-844

Gupta, O.K.; Ravindran, A. 1983: Nonlinear integer programming and discrete optimization. *J. Mech. Trans. Autom. Des. ASME* **105**, 160-164

Hadley, G. 1962: *Linear programming.* Reading, MA: Addison-Wesley

Hager, K.; Balling, R.J.1988: New approach for discrete structural optimization. *J. Struct. Engng. ASCE* **114**, 1120-1134

Hajela, P. 1989: Genetic search – an approach to the nonconvex optimization problem. *Proc. 30th AIAA/ASME/ASCE/ASHS Structures, Structural Materials and Dynamics Conf.* (held in Mobile, Alabama), pp. 165-175

Hajela, P.; Shih, C.-J. 1990: Multiobjective optimum design in mixed integer and discrete design variable problems. *AIAA J.* **28**, 670-675

Hua, H.M. 1983: Optimization of structures with discrete-size elements. *Comp. & Struct.* **17**, 327-333

John, K.V.; Ramakrishnan, C.V.; Sharma, K.G. 1988: Optimum design of trusses from available sections-use of sequential linear programming with branch and bound algorithm. *Engng. Optim.* **13**, 119-145

Jonsson, Ö.; Larsson, T. 1990: Lagrangean relaxation and subgradient optimization applied to optimal design with discrete sizing. *Eng. Opt.* **16**, 221-233

Kelahan, R.C.; Gaddy, J.L. 1978: Application of the adaptive random search to discrete and mixed integer optimization. *Int. J. Num. Meth. Eng.* **13**, 119-145

Kelly, J.E. 1969: The cutting plane method for solving convex programs. *J. SIAM* **8**, 702-712

Kincaid, R.K.; Padula, S.L. 1990: Minimizing distortion and internal forces in truss structures by simulated annealing. *Proc. 31st AIAA/ASME/ASCE/AHS/ASC Structures, Structural Materials and Dynamics Conf.* (held in Long Beach, California), Paper No, AIAA-90-1095-CP, pp. 327-333

Kirkpatrick, S.; Gelatt, C.D.; Vecchi, M.P. 1983: Optimization by simulated annealing. *Science* **220**, 671-680

Land, A.M.; Doig, A.G. 1960: An automatic method of solving discrete programming problems. *Esconometrica* **28**, 497-520

Lawler, E.L.; Bell, M.D. 1966: A method for solving discrete optimization problems. *Operations Research* **14**, 1098-1112

Lee, T.W.; Freudenstein, F. 1976: Heuristic combinatorial optimization in the kinematic design of mechanisms. *ASME J. Engineering for Industry* **4**, 1277-1284

Lin, C.-Y.; Hajela, P. 1992: Genetic algorithms in optimization problems with discrete and integer design variables. *Engng. Optim.* **19**, 309-327

Lin, S.; Kernighan, B.W. 1973: An effective heuristic algorithm for solving mixed-discrete nonlinear design optimization problems. *Operations Research* **21**, 498-516

Loh, H.T.; Papalambros, P.Y. 1991: A sequential linearization approach for solving mixed-discrete nonlinear design optimization. *J. Mech. Des. ASME* **113**, 325-334

Loh, H.T.; Papalambros, P.Y. 1991: Computational implementaion and tests for solving mixed-discrete nonlinear design optimization problems. *J. Mech. Des. ASME* **113**, 335-345

Lucidi, S.; Piccioni, M. 1989: Random tunneling by means of acceptance-rejection sampling for global optimization. *J. Optim. Theory Appl.* **62**, 255-277

Luenberger, D. 1984: *Linear and nonlinear programming.* Reading, MA: Addison-Wesley

May, S.A.; Balling, R.J. 1991: Strategies which permit multiple discrete section properties per member in 3D frameworks. In: Ural, O.; Wang, T.-L. (eds.) *Electronic computation*, pp. 189-196. New York: ASCE

May, S.A.; Balling, R.J. 1992: A filtered simulated annealing strategy for discrete optimization of 3D steel frameworks. *Struct. Optim.* **4**, 142-148

Mesquita, L.; Kamat, M. 1987: Optimization of stiffened laminated composite plates with frequency constraints. *Engng. Optim.* **11**, 77-88

Metropolis, N.; Rosenbluth, A.W.; Rosenbluth, M.N.; Teller, A.H.; Teller, E. 1953: Equations of state calculations by fast computing machines. *J. Chemical Physics* **21**, 1087-1092

Ming-Zhu, D. 1986: An improved Templeman's algorithm for the optimum design of trusses with discrete member sizes. *Eng. Opt.* **9**, 303-312

Minoux, M. 1986: *Mathematical programming theory and algorithms.* New York: John-Wiley & Sons

Mottl, J. 1992: Excavator optimization using the "voting method". *Comp. Meth. Appl. Mech. Engng.* **98**, 227-250

Olsen, G.R.; Vanderplaats, G.N. 1989: Method for nonlinear optimization with discrete design variables. *AIAA J.* **27**, 1584-1589

Pyrz, M. 1990: Discrete optimization of geometrically nonlinear truss structures under stability constraints. *Struct. Optim.* **2**, 125-131

Reinschmidt, K. 1971: Discrete structural optimization. *J. Struct. Div. ASCE* **94**, 133-156

Reiter, S.; Rice, D.B. 1966: Discrete optimizing solution procedures for linear and nonlinear integer programming problems. *Management Science* **12**, 829-850

Reiter, S.; Sherman, G. 1965: Discrete programming. *J. Society for Industrial and Applied Mathematics* **13**, 864-889

Ringertz, U.T. 1988: On methods for discrete structural optimization. *Engng. Optim.* **13**, 47-64

Salajegheh, E.; Vanderplaats, G.N. 1993: Optimum design of trusses with sizing and shape variables. *Struct. Optim.* **6**, 79-85

Salkin, H.M. 1975: *Integer programming.* Reading, MA: Addison-Wesley

Salmon, C.G.; Johnson, J.E. 1979: *Steel structures design and behavior.* New York: Harper and Row

Sandgren, E. 1990a: Nonlinear integer and discrete programming in mechanical design optimization. *ASME J. Mech. Des.* **112**, 223-229

Sandgren, E. 1990b: Nonlinear integer and discrete programming for topological decision making in engineering design. *ASME J. Mech. Des.* **112**, 118-122

Schmit, L.; Fleury, C. 1980: Discrete–continuous variable structural synthesis using dual methods. *AIAA J.* **18**, 1515-1524

Schrage, L. 1983: *LINDO – linear interactive discrete optimizer.* Chicago: University of Chicago

Sepulveda, A.; Cassis, J. 1986: An efficient algorithm for the optimum design of trusses with discrete variables. *Int. J. Num. Meth. Eng.* **23**, 1111-1130

Shin, D. D.K.; Gurdal, Z.; Griffin, O.H., Jr. 1990: A penalty approach for nonlinear optimization with discrete variables. *Int. J. Num. Meth. Eng.* **23**, 1111-1130

Siddall, J.N. 1982: *Optimal engineering design.* New York: Marcel Decker

Sugimoto, H. 1992: Discrete optimization of truss structures and genetic algorithms. In: Choi, C.-K.; Sugimoto, H.; Yun, C.-B. (eds.) *Proc. Korea-Japan Joint Seminar on Structural Optimization* (held in Seoul, Korea), pp. 1-10. Computational Structural Engineering Institute of Korea, Seoul

Templeman, A.B.; Yates, D.F. 1983a: A linear programming approach to the discrete optimum design of trusses. In: Eschenauer, H.; Olhoff, N. (eds.) *Optimization methods in structural design*, pp. 133-139. Mannheim: BI Wissenschaftsverlag

Templeman, A.B.; Yates, D.F. 1983b: A segmental method for the discrete optimum design of structures. *Engng. Optim.* **6**, 145-155

Toakley, A.R. 1968: Optimum design using available sections. *ASCE J. Struct. Div. ASCE* **34**, 1219-1241

Tseng, C.H.; Wang, L.W.; Ling, S.F. 1992: A numerical study of the branch-and-bound method in structural optimization. *Technical Report*, Department of Mechanical Engineering, National Chiao Tung University, Taiwan

Vanderplaats, G.N.; Thanedar, P.B. 1991: A survey of discrete variable optimization for structural design. In: Ural, O.; Wang, T.-L. (eds.) *Electronic computation*, pp. 173-180. New York: ASCE

Yuan, K.Y.; Liang, C.C.; Ma, Y.C. 1990: The estimation of the accuracy and efficiency of the backtrack programming method for discrete-variable structural optimization problems. *Comp. & Struct.* **36**, 211-222

Received July 22, 1993
Revised manuscript received April 6, 1994