ENHANCING BRANCH-AND-BOUND METHOD FOR STRUCTURAL OPTIMIZATION

By C. H. Tseng,[1] L. W. Wang,[2] and S. F. Ling[3]

ABSTRACT: The branch-and-bound method was originally developed to cope with difficulties caused by discontinuous design variables in linear programming. When the branch-and-bound method is applied to solve nonlinear programming (NLP) problems with a large number of mixed discontinuous and continuous design variables, the slow rate of convergence becomes a major drawback of the method. In this study, a number of enhancements are proposed to speed up the rate of convergence of the conventional branch-and-bound algorithm. Three NLP in the form of truss-design examples are tested to compare the capabilities and efficiency of the proposed enhancements. It is shown that of the five criteria for arranging the order in which the design variables are branched, the criterion of maximum cost difference dramatically reduces the number of branch nodes, thereby reducing the total number of continuous-optimization runs executed. Moreover, neighboring search, a branching procedure restricted in the neighborhood of the continuous optimum, is proven to be effective in speeding up the convergence. Investigation also shows that branching several design variables simultaneously is not as efficient as sequentially branching one variable at a time. The proposed enhancements are incorporated along with a sequential quadratic programming algorithm into a software package that is shown to be very useful in the optimal design of engineering structures.

INTRODUCTION

As computer technology advances, optimization is becoming a powerful tool in engineering design. In conventional optimization algorithms, complications such as nonlinear functions, multiple objective functions, and a large number of design variables have been dealt with successfully [see, for example, Arora (1989)]. Although continuous design variables are usually assumed in conventional algorithms, discontinuous design variables are frequently encountered in practical applications. Examples are the size of a set of standard components (discrete), the number of teeth of a gear (integer), or a choice between different design options (zero-one). The capability to deal with discontinuous design variables via conventional nonlinear-programming (NLP) problem algorithms would be a major expansion of the applications of structural optimization.

To make use of well-established continuous-optimization algorithms, most discrete-optimization techniques are based on the assumption of transforming the discontinuous solution space into multiple continuous solution subspaces. The optimization problems in each of these continuous subspaces are solved sequentially by imposing constraints on discreteness of the design variables. The optimal discrete solution is chosen from among the continuous solutions obtained (Gochet and Smeers 1979). In addition to application cases of the branch-and-bound method that have been reported in references (Siddall 1982; Gupta and Ravindran 1981; Lee 1983), a general algorithm for solving NLP problems with integer and discrete design variables is presented by Sandgren (1990). However, there is a major obstacle to applying this algorithm to problems with a large number of discontinuous design variables: it requires too many executions of the continuous-optimization scheme and thus is very time consuming (Tseng and Wang 1989). Hager and Balling (1988) have solved the discrete-variable optimization problem for the design of steel frames. The problem is first converted into a linear programming (LP) problem after the continuous-optimum solution is obtained. Then the LP problem is solved using the branch-and-bound method in the neighborhood of the continuous optimum. This approach enables the branch-and-bound method to solve large-scale discrete-optimization problems effectively. Generally speaking, a survey of the literature (Vanderplaats and Thanedar 1991) shows that little numerical experience in using the branch-and-bound method has been reported.

In this study, a number of enhancements to the conventional branch-and-bound method are proposed to reduce the number of executions of the continuous-optimization scheme. The effectiveness of the following are investigated: (1) Selected branching order of nodes; (2) selected branching order of design variables; (3) neighboring search in the subspace near the continuous optimum; and (4) modified branching algorithm itself to improve the efficiency of the branch-and-bound method. The objective is to maximize the efficiency of using the branch-and-bound method in solving large-scale discrete-optimization problems. The proposed enhancement are coded in IDESNC to be readily incorporated with a sequential quadratic programming (SQP) algorithm in an optimization program IDESIGN3.51 [a modified version of the package IDESIGN3.5 used for solving continuous-optimization problems (Arora 1989)]. To evaluate and compare the numerical performance of the various enhancements proposed, we developed the software IDESNC to solve three typical truss-design problems.

METHODOLOGY

Branching Order of Nodes

To locate a discrete optimum solution using continuous-optimization schemes, the branch-and-bound method repeatedly deletes portions of the original design space that do not contain allowable values of the discontinuous design variables. This procedure is called "branching." First of all, the problem is solved by using a continuous-optimization algorithm. If the continuous optimum occurs at an allowable discrete value for each of the design variables, the branching procedure should be stopped. Otherwise, the procedure should be continued. As illustrated in Fig. 1, the original design space

[1]Prof., Dept. of Mech. Engrg., Nat. Chiao Tung Univ., Hsinchu, Taiwan, Republic of China.

[2]Grad. Student, Dept. of Mech. Engrg., Nat. Chiao Tung Univ., Hsinchu, Taiwan, Republic of China.

[3]Assoc. Prof., School of Mech. and Production Engrg., Nanyang Technol. Univ., Singapore.

Note. Associate Editor: Jasbir S. Arora. Discussion open until October 1, 1995. To extend the closing date one month, a written request must be filed with the ASCE Manager of Journals. The manuscript for this paper was submitted for review and possible publication on June 17, 1992. This paper is part of the ***Journal of Structural Engineering***, Vol. 121, No. 5, May, 1995. ©ASCE, ISSN 0733-9445/95/0005-0831–0837/ $2.00 + $.25 per page. Paper No. 4252.

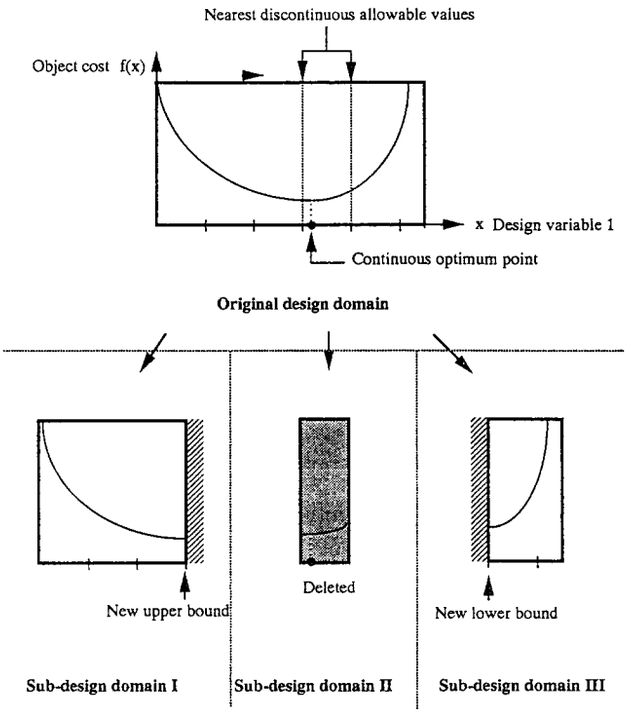

FIG. 1. Conceptual Layout of Branching Procedures

is divided into three subspaces with the allowable discontinuous design values nearest the continuous-optimum solution as the upper and the lower bound of the left and the right subspaces, respectively. The center subspace, including the original continuous solution but excluding the feasible discontinuous solution, is dropped. In each of the remaining two subspaces, the minimum cost may occur at a value of the design variable other than the newly assigned bound. A conventional continuous-optimization scheme has to be used again to find the optimum for each of the two remaining subspaces. If the new optimum does not occur at an allowable discrete value of the design variable in either of the subspaces, the foregoing branching procedure has to be repeated in each of the subspaces until a feasible optimal design is located. In the process, the combination of design subspaces always includes all feasible discrete design variables, no matter how many levels of branching have been done. In addition, as the design space of the subproblem grows smaller, it becomes easier to identify the continuous solution in a subspace.

The "tree" of the branch-and-bound method, i.e., a diagram of the branching, is shown in Fig. 2. Each design subspace is depicted as a "node." In principle, a discrete solution can be found if an exhaustive search of the tree is made. Conceptually, the exhaustive search can be either depth first or breadth first, as illustrated in Figs. 2(a and b), respectively. In a depth-first search, searching proceeds downward as far as possible before the search of another branch begins. In contrast, a breadth-first search proceeds one level at a time: one level is searched completely before descending to the next lower level. Both searching methods involve a large number of continuous optimization and thus are very time-consuming for the computer. Choosing an appropriate route in the tree so that the solution is reached more directly would drastically reduce the computing time required for the branch-and-bound method. Such a search route is depicted in Fig. 2(c).

In discrete optimization, the minimum cost in the original continuous design space is always less than or equal to that in the design subspaces that have been branched from the original space. This fact provides a guideline about when to

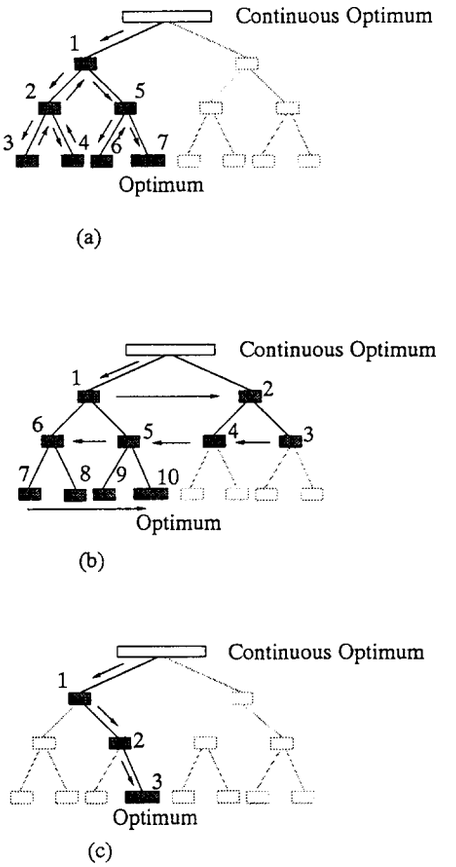

FIG. 2. Historical Diagrams of Searching Methods: (a) Depth-First Search; (b) Breadth-First Search; (c) Best-First Search

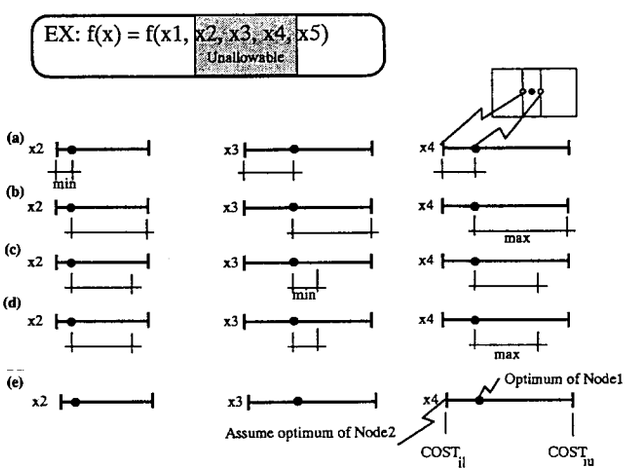

FIG. 3. Branching Order of Design Variables: (a) Minimum Clearance; (b) Maximum Clearance; (c) Minimum Clearance Difference; (d) Maximum Clearance Difference; (e) Maximum Cost Difference

stop branching further in depth. If a feasible discontinuous solution is obtained in the process of branching, the corresponding cost-function value can be taken as a bound. Any other design subspace that possesses a continuous minimum cost larger than this bound need not be further branched, because further branching will only generate higher costs. This strategy is called "bounding," and can be used to choose a branching route intelligently so as to avoid complete and impartial searching through the tree. The idea of bounding is useful only when the continuous-optimization scheme employed yields the global minimum in each of the design subspaces. Theoretically, this is not always the case, because

there is no conventional optimization scheme that guarantees convergence to the global minimum except for the convex problems. Nevertheless, in the branch-and-bound method, the good news is that the possibility of converging to a global minimum increases enormously in practice because the same optimization problems are solved repeatedly in different design subspaces with different initial guesses of design variables.

Branching Order of Design Variables

The comments in last section concentrated on the branching order of nodes for a single design variable. When there are a large number of design variables, the order of design variables to be branched at the same node influences the efficiency of the method largely. In the following, we propose five different criteria to determine the priority in which the design variables are branched (Fig. 3).

1. *Minimum Clearance*: For each design variable, we calculate the clearance between the nonallowable optimum point and the nearest allowable design value. The design variable possessing the minimum clearance is selected to be branched in the next step. The criterion is as follows:

$$\Delta x_i = \min(|x_i - X_{il}|, |x_i - X_{iu}|) \tag{1}$$

$$x_{obj} = \min(\Delta x_1, \Delta x_2, \ldots, \Delta x_n) \tag{2}$$

where x_i = optimum value of the ith design variable; X_{il} and X_{iu} = lower and upper allowable values of the ith design variable, respectively; Δx_i = smallest clearance of design variable i; and x_{obj} = design variable to be branched.

2. *Maximum Clearance*: The criterion is the same as just described except that the design variable with the maximum clearance is branched first, as follows:

$$\Delta x_i = \max(|x_i - X_{il}|, |x_i - X_{iu}|) \tag{3}$$

$$x_{obj} = \max(\Delta x_1, \Delta x_2, \ldots, \Delta x_n) \tag{4}$$

3. *Minimum Clearance Difference*: This criterion is the difference between the clearances of the nonallowable optimum point and the nearest lower and upper allowable values of each design variable. The design variable with the minimum clearance difference is taken as the branch object, as follows:

$$\Delta x_i = ||x_i - X_{il}| - |x_i - X_{iu}|| \tag{5}$$

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

4. *Maximum Clearance Difference*: This is the same as criterion 3 except that the design variable with the maximum clearance difference is taken as the branch object, as follows:

$$\Delta x_i = ||x_i - X_{il}| - |x_i - X_{iu}|| \tag{7}$$

$$x_{obj} = \max(\Delta x_1, \Delta x_2, \ldots, \Delta x_n) \tag{8}$$

5. *Maximum Cost Difference*: At a node, we fix the values of all design variables except one, and then evaluate the costs at the lower and upper bounds for the remaining variable. The design variable with the largest difference between these two costs is the branch object, as follows:

$$\text{Cost}_{il} = f(x_1, x_2, \ldots, X_{il}, \ldots, x_n) \tag{9}$$

$$\text{Cost}_{iu} = f(x_1, x_2, \ldots, X_{iu}, \ldots, x_n) \tag{10}$$

$$\Delta\text{Cost}_i = |\text{Cost}_{il} - \text{Cost}_{iu}| \tag{11}$$

$$x_{obj} = \max(\Delta\text{Cost}_1, \Delta\text{Cost}_2, \ldots, \Delta\text{Cost}_n) \tag{12}$$

where Cost_{il} and Cost_{iu} = cost value of the lower and upper bounds, respectively, of ith design variable at a node, and

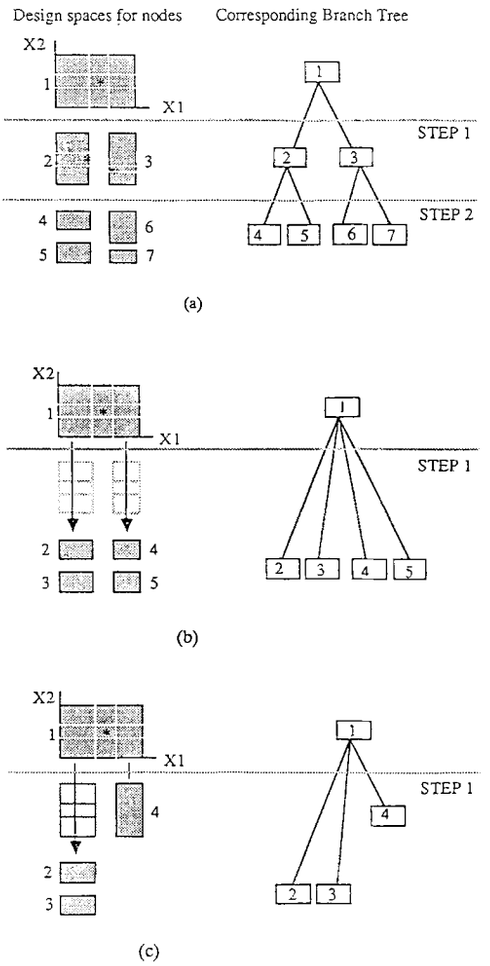

FIG. 4. Conceptual Layout of Branching Algorithms: (a) Single Branch; (b) Multiple Branch; (c) Unbalanced Branch

ΔCost_i = difference between the two costs. This criterion has been used in Hager and Balling's 1988 paper.

The linear sensitivity coefficient, $\partial f/\partial x_i$, of the cost function at the optimum solution in the last higher level of the branching tree can be used to estimate the cost difference between the lower and upper bounds of the ith design variable using the following equation:

$$\Delta\text{Cost}_i = \left|\frac{\partial f}{\partial x_i}(X_{iu} - X_{il})\right| \tag{13}$$

If this linear approximation is used to calculate ΔCost_i, it is not necessary to use (9) and (10) to call the cost-function routine to evaluate the cost values. This approximation thus greatly reduces the number of function calls.

In the first four criteria, the clearance between a continuous optimal point and its nearest discontinuous design point is taken as the criterion. By implication, the sensitivities of the cost function to all design variables are the same. This assumption makes evaluation and judgment fast and easy at the expense of accuracy. It is clear that the last criterion improves accuracy.

Neighboring Search

After a continuous-optimum solution is obtained at the first node, the algorithm searches through the subspace between NB (a number of discrete value) larger and NB smaller discrete values of each design variable out from the optimum to look for the discrete minimum. The value of NB can be chosen

J. Struct. Eng. 1995.121:831-837.

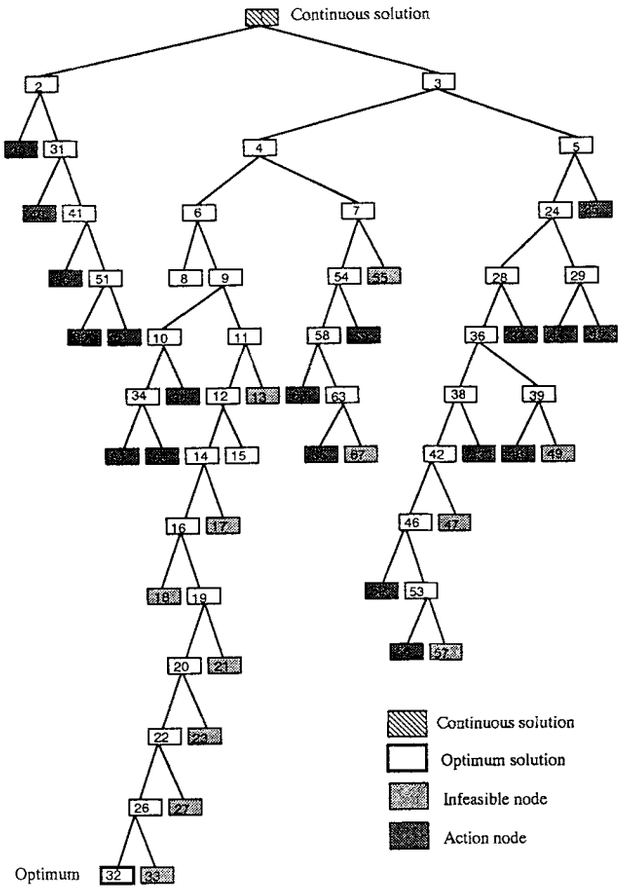

FIG. 5. Historical Diagram of Branching of 10-Bar–Truss Example

as one, two, or larger. For example, $NB = 1$ means that one bigger and one smaller discrete allowable values for each design variable in the neighborhood of the continuous optimum are taken as the subspace for searching in the remaining procedure. If some design variables of the continuous optimum are right at the lowermost (or uppermost) discrete values then the right (or left) subspace for NB larger (or smaller) discrete values for that design variable are taken as neighbors. If NB = all in present paper, it means that all discrete values for each design variable in the original design space are used for the branching procedure.

Branching Algorithms

In branching algorithms; one node is always branched into two nodes in the next level in the tree, as shown in Fig. 4(a). If several design variables are dealt with simultaneously, the next level can be of more than two nodes. In this study, the following two new branching algorithms are proposed to improve the efficiency of the branch-and-bound method.

1. *Multiple Branching*: Fig. 4(b) for Multi-2 demonstrates the idea of multiple branching using the case of two discontinuous design variables. For each of the variables, two subspaces are generated. Therefore, if the number of simultaneously treated design variables is n for Multi-n, the number of nodes in one branching will be 2^n. Comparing Figs. 4(a and b) show that to achieve the same state of design subspaces in the next lower level of a tree, single-branching requires two steps of branching and 6 nodes in total produced, while the multibranching algorithm needs one step only with 4 nodes produced. Thus, in principle, multiple branching should be able to reduce the computing power required.

2. *Unbalanced Branching*: A conceptual layout of this algorithm is illustrated in Fig. 4(c). This layout represents a compromise of the multiple-branching algorithm that avoids the disadvantages of complete and impartial branching. The algorithm can be thought of as a combination of the first and second steps of the single branching algorithm with one of the two nodes selected according to the concept of bounding.

NUMERICAL EXAMPLES

To investigate the performance of the foregoing methods, the branch-and-bound method with the proposed enhancements is incorporated with a SQP algorithm to solve three truss-diagram problems each with seven, 10, and 28 design variables, respectively. All three problems has been solved by Haug and Arora (1979), Thanedar et al. (1986), and Arora and Tseng (1988) as continuous-optimization problems. The finite-element method is used for structural analysis. A hybrid design sensitivity method (Tseng and Kao 1989) that combines the advantages of the direct differentation method (DDM) and the adjoint variable method (AVM) is used to calculate the sensitivity coefficients. All the computations are done on an HP 835 workstation.

1. *10-Member Cantilever Truss*: The structure and its loading is fully described in the case II of Example 4.5 by Haug and Arora (1979, page 242). The objective is to minimize the weight of the truss. The cross-sectional areas are taken as design variables. Nineteen constraints on stress, displacement, member buckling, and fundamental vibration frequency are imposed. The design formulation of the discrete-optimization problem is the same as that in the continuous case except the cross-sectional area of the truss members needs to be selected from the following set of discrete values: 1.62, 1.80, 1.99, 2.13, 2.38, 2.62, 2.63, 2.88, 2.93, 3.09, 3.13, 3.38, 3.47, 3.55, 3.63, 3.84, 3.87, 3.88, 4.18, 4.22, 4.49, 4.59, 4.80, 4.97, 5.12, 5.74, 7.22, 7.97, 11.5, 13.5, 13.9, 14.2, 15.5, 16.0, 16.9, 18.8, 19.9, 22.0, 22.9, 26.5, 30.0, and 33.5. In practice, these discrete values may be those prescribed in the construction codes, e.g., the American Institute of Steel Construction (AISC) code.

Five criteria for the branching order of the design variables [(1) minimum clearance; (2) maximum clearance; (3) minimum clearance difference; (4) maximum clearance difference; and (5) maximum cost difference], four neighboring searches (NB = All, 3, 2, and 1), and the three branching algorithms (single, multiple, and unbalanced) are tested. Table 1 shows the results of these studies and the comparable results given by Arora and Tseng (1988). The discontinous-optimum solution found by IDESNC is 0.9% better than that reported by Arora and Tseng (1988), which was only a guess solution obtained by the interactive capability of IDESIGN3.5. Fig. 5 is the tree diagram when the maximum-cost-difference criterion with NB = all is adopted. Altogether, there are 67 nodes and 442 function evaluations, and the cost-function value increases 0.33% as compared to the continuous optimum. Eleven nodes are deleted because they contained no feasible variable, and the first allowable discrete solution was obtained at node 32. The optimum cost value at this node was set as the bound for the nodes not yet branched, because they may possess optimum cost lower than the bound. The branching of each of the remaining nodes was terminated once an optimum was found that was larger than the established bound. This case took the central processing unit (CPU) approximately 20 times longer than the time required by the corresponding continuous optimization.

The results of the single branching and the maximum-cost-difference criterion with neighboring search for the number of discrete values (NB = 1, 2, 3, and all) are also shown in Table 1. The number of nodes and the function calls and the CPU time decreased when a smaller NB number was used.

TABLE 1. Summary of Results for 10-Bar–Truss Design Example

Branch order of design variables	SINGLE BRANCH AND BOUND								MULTI BRANCH AND BOUND			
	(1)	(2)	(3)	(4)	(5)				Multi-2	Multi-3	Multi-4	Un-bala
					NB = all	*NB* = 3	*NB* = 2	*NB* = 1				
Number of nodes	471	101	275	89	67	53	43	5	105	205	121	99
Function calls	3,725	682	2,084	598	442	360	320	51	963	1,727	1,113	1,076
Weight	5,431	5,414	5,414	5,415	5,414	5,414	5,414	5,489	5,415	5,415	5,415	5,415
CPU time (sec)	211.5	40.1	129.2	40.0	29.5	26.1	23.2	3.8	54.9	102.9	68.5	48.6

Optimum Solutions

Continuous design variables	1 28.279	2 1.620	3 27.261	4 13.727	5 1.620	6 4.003	7 13.595	8 17.544	9 19.130	10 1.62
Function calls Weight CPU time (sec)	22 5,396.530 1.4									
Discontinuous discrete design variables[a]	1 30.00	2 2.62	3 26.50	4 13.90	5 1.62	6 2.62	7 13.50	8 18.80	9 18.80	10 1.62
Weight Weight increase (%)	5,465.430 1.27									
Discontinuous discrete design variables[b]	1 30.00	2 1.80	3 26.50	4 14.20	5 1.62	6 3.84	7 13.90	8 16.90	9 18.80	10 1.62
Weight Weight increase (%) CPU time (sec)	5,414.256 0.328 29.5									

Note: 10 (discrete) design variables; 19 constraints; and 420 (42 × 10) discrete values.
[a]Arora and Tseng (1988).
[b]IDESNC; *NB* = all.

TABLE 2. Summary of Results for 25-Member Transmission-Tower Design Example

Branch order of design variables	SINGLE BRANCH AND BOUND								MULTI BRANCH AND BOUND			
	(1)	(2)	(3)	(4)	(5)				Multi-2	Multi-3	Multi-4	Un-bala
					NB = all	*NB* = 3	*NB* = 2	*NB* = 1				
Number of nodes	25	27	25	19	15	15	13	7	25	23	23	24
Function calls	121	133	121	122	70	82	87	50	161	204	272	179
Weight	720.4	722.8	720.4	722.8	720.4	720.4	720.4	722.8	720.4	720.4	720.4	720.4
CPU time (sec)	26.6	28.1	26.7	25.0	15.3	17.6	18.8	10.2	33.3	39.3	51.3	30.1

Optimum Solutions

Continuous design variables	1 1.62	2 2.147	3 2.334	4 1.62	5 1.62	6 1.974	7 3.431
Function calls Weight CPU time (sec)	13 716.132 0.8						
Discontinuous discrete design variables[a]	1 1.620	2 2.130	3 2.380	4 1.620	5 1.620	6 1.990	7 3.470
Weight Weight increase (%) CPU time (sec)	720.429 0.600 15.3						

Note: 7 (discrete) design variables; 87 constraints; and 294 (42 × 7) discrete values.
[a]IDESNC; *NB* = all.

The final value of the cost function increased when *NB* was a small number because the subspace of neighbors about the continuous optimum was small. The neighboring search makes the process more efficient but not very accurate.

2. *25-Bar Tower*: The geometry and dimensions of a 25-member transmission tower are given by Haug and Arora (1979, page 245). Because of the symmetry of the structure and the use of the design-variable linking technique, seven design variables are used to represent the cross-sectional areas of the 25 truss members. There are 87 constraints imposed. The five criteria for the branching order of the design variables, four neighboring searches, and the three branching algorithms are also tested. Table 2 shows the results of these investigations. The number of nodes decreased as smaller *NB*s were used. However, the number of function calls and the CPU time increased for the cases when *NB* = 2 and 3 compared with that where *NB* = all. Checking the procedure in detail shows that one of the branching nodes needed a larger number of function calls and iterations to obtain convergence in the SQP algorithm.

3. *200-Member Structure*: To demonstrate the use of the enhanced branch-and-bound method for large-scale problems, a structure with 200 members, 77 joints, 150 degrees of freedom, and three independent loading conditions was selected as an example. The detailed design data of the structure can be found in Haug and Arora (1979, page 250). Design-variable linking is used to represent the cross-sectional areas for 200 members in 29 design variables. The optimi-

J. Struct. Eng. 1995.121:831-837.

TABLE 3. Summary of Results for 200-Bar 29–Design Variables Truss-Design Example

Branch order of design variables	SINGLE BRANCH AND BOUND							
	(1)	(2)	(3)	(4)	(5)			
					NB = all	*NB* = 3	*NB* = 2	*NB* = 1
Number of nodes	3,000[a]	3,000[a]	3,000[a]	187	131	59	49	13
Function calls	19,805[a]	23,224[a]	19,368[a]	1,317	859	512	347	59
Weight	+	+	+	34,585	34,585	34,585	34,593	34,698
CPU time (sec)	124,325[a]	147,100[a]	124,924[a]	8,995.5	6,216.8	3,387.2	2,267.8	464.5
Optimum Solutions								
Continuous								
Design variables	1.620	1.620	1.620	1.620	1.620	1.620	1.620	2.278	1620	3.278
	1.620	1.620	4.837	1.620	5.837	1.620	1.620	7.860	1.620	8.861
	1.620	1.620	11.997	1.620	13.997	2.364	4.048	8.312	16.827	
Function calls	16									
Weight	32,974.725									
CPU time (sec)	174.8									
Discontinuous[b]										
Discrete design variables[b]	1.620	1.620	1.620	1.620	1.620	1.620	1.620	2.380	1.620	3.380
	1.620	1.620	4.800	1.620	5.740	1.620	1.620	7.970	1.620	11.500
	1.620	1.620	11.500	1.620	13.500	2.130	3.840	11.500	15.500	
Weight	34,585.147									
Weight increase (%)	4.884									
CPU time (sec)	6,216.8									

Note: 29 (discrete) design variables; 1,600 constraints; and 1,218 (42 × 29) discrete values.
[a]IDESNC; *NB* = all.

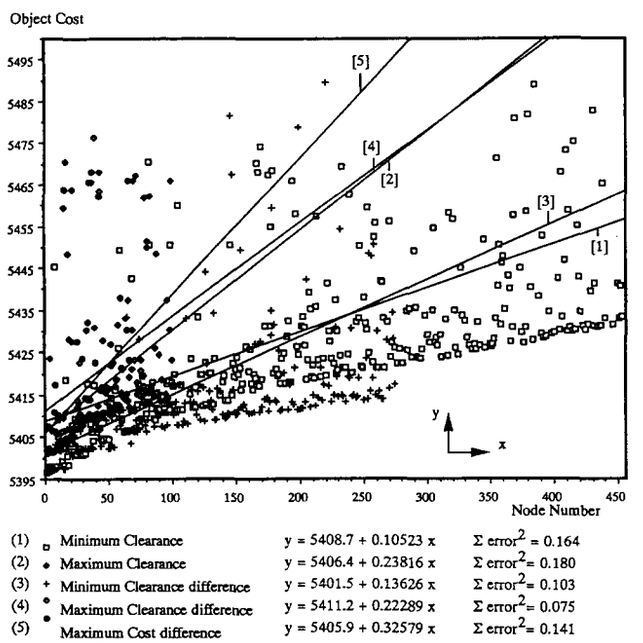

FIG. 6. Cost-increment Diagram of 10-Bar–Truss Example

zation problems has 600 constraints in terms of member stress, nodal displacement, buckling, and the fundamental vibration frequency. Explicit design-variable bounds are also imposed. The formulation of this optimization is the same as that given in Haug and Arora (1979) and Arora and Tseng (1988) except that the design variables are discrete.

The continuous and discrete solutions are both listed in Table 3. The five criteria for the branching order of the design variables are again tested. When minimum clearance, maximum clearance, and minimum clearance difference are used as the criteria to determine the order of the branching design variables (cases 1 to 3), the solutions did not converge until the number of generated nodes reached 3,000. For the maximum clearance difference and maximum cost difference criteria (cases 4 and 5), however, only 187 nodes and 131 nodes, respectively, are generated in the process and the cost function increases by only 4.9%. Although a large amount of CPU time is needed, the enhanced branch-and-bound method provides a discrete-optimum solution.

The trend for the neighboring search is the same as that in the 10-bar–truss example. The number of nodes and function calls and the CPU time decreased tremendously, and the final value of cost function increases slightly for the case in which *NB* = 1 compared with the case in which *NB* = all.

COMMENTS

Branching Order of Design Variables

All five criteria proposed for determining the order in which the design variables are branched are tested for both the 10-member cantilever truss and the 25-bar–tower examples to compare their effectiveness in improving the efficiency of the branch-and-bound method. The results are shown in Tables 1 and 2. The rate of convergence and number of nodes created are significantly different. The criterion of maximum cost different performs best. This branching order promotes growth of the tree in depth rather than breadth. In Fig. 6, the cost-function values of the 10-member cantilever truss are plotted against the number of nodes for the five proposed criteria. The slopes of the best-fit lines indicate the rate of convergence of each criterion. The obvious difference in the slope of these lines indicates that the order of the branching design variable is the dominant criterion in determining the rate of convergence of the branch-and-bound method. As shown in Fig. 6, the criterion of maximum cost difference has the steepest slope and thus converges fastest.

Neighboring Search

If a discrete-optimum solution is near the continuous-optimum solution or if the optimization problem is not highly nonlinear, *NB* allowable discrete values for each design variable in the neighborhood of the continuous-optimum solu-

tion can be sought to obtain the discrete-optimum solution very efficiently and accurately. If the problem does not fall into one of these categories, the value of NB must be increased to obtain a feasible solution, and the neighboring search becomes less efficient. In the results of the 25-bar-tower example, the number of function calls increases when the neighboring search is employed simply because a larger number of function calls and iterations is needed in the SQP algorithm. In practical usage for large-scale problems, it suggests the use of a smaller value of NB to obtain the value of the objective function in the beginning. Then a user can increase the value of NB to obtain the new value of the objective function. If the new value of the objective function is converged, it can stop process for increasing the value of NB. However, the value of NB = all is still optional for the user to obtain the accurate solution if CPU time is accessible.

Branching Algorithms

The underlying concept of using alternative branching algorithms is to eliminate the middle nodes of a tree by allowing a branch of more than one design variable at a time. The proposed methods are applied to the 10-member cantilever truss and the 25-member transmission tower; the results are shown in Tables 1 and 2. The efficiency of the branch-and-bound method has not improved as expected.

To understand why, let us refer to Figs. 3(a and b) again. In multiple branching, one step of branching gives four nodes; however, these are different from and not as good as the four obtained by the two steps of the single branching. This is because the two algorithms use different information to select branching objects. In step two of single branching, new information on the nodes obtained in the first step is used; in multiple branching, only the original information on the nodes is available to select the two branch objects in a single step.

In single branching, sometimes one of the nodes generated in the first step will not be further branched in the second step, as illustrated in Fig. 3(a). In every step, one of the nodes may be abandoned, for reasons discussed previously. In multiple branching, however, $2n$ nodes are generated directly without providing the opportunity to check whether some of the nodes should be dropped. More nodes than necessary may be generated, thereby slowing down the rate of convergence of the solution process.

Unbalanced branching is proposed to overcome the aforementioned disadvantages of multiple branching. Test results show that it does improve the rate of convergence. Nevertheless unbalanced branching cannot totally overcome the disadvantages of multiple branching, and the solution speed of this algorithm is slower than that of the single branching algorithm.

CONCLUSION

The following conclusions may be drawn from the numerical study presented here.

First, maximum cost difference, proposed by Hager and Balling (1988), is the most effective criterion among the five proposed criteria for determining the order of branching of the design variables.

Second, a modified branch-and-bound method with the capability of performing neighboring search is efficient but less accurate. The modified method is more suitable for solving large-scale engineering problems.

Third, single branch algorithm, which simplifies the branch tree the most, is more effective than the multiple and unbalanced branching algorithms for discrete optimization.

The software IDESNC, which combines the enhancements with the SQP optimization scheme, can be effective and efficient in solving three structural-optimization problems involving discrete design variables. In the future, the problems of different scales (e.g., a larger number of design variables) and different types (mixed, discrete, and continuous) should be further investigated to make the branch-and-bound method more useful for solving engineering problems.

ACKNOWLEDGMENTS

The research reported in this paper was supported under a project sponsored by the National Science Council, Taiwan, Republic of China, grant number NSC78-0401-E009-12.

APPENDIX I. REFERENCES

Arora, J. S. (1989). *Introduction to optimal design*. McGraw-Hill, New York, N.Y.

Arora, J. S., and Tseng, C. H. (1987). "IDESIGN: User's manual version 3.5." *Tech. Rep. No. OD-87.1*, Optimal Des. Lab., Coll. of Engrg., Univ. of Iowa, Iowa City, Iowa.

Arora, J. S., and Tseng, C. H. (1988). "Interactive design optimization." *Engrg. Optimization*, Vol. 13, 173–188.

Gochet, W., and Smeers, Y. (1979). "A branch-and-bound method for reverse geometric programming." *Operations Res.*, 27(5), 982–996.

Gupta, O. K., and Ravindran, A. (1981). "Nonlinear integer programming and discrete optimization." *Progress in engineering optimization*, R. W. Mayne and K. M. Ragsdell, eds., ASME, New York, N.Y. 27–32.

Hager, K., and Balling, R. (1988). "New approach for discrete structural optimization." *J. Struct. Engrg.*, ASCE, 114(5), 1120–1134.

Haug, E. J., and Arora, J. S. (1979). *Applied optimal design: mechanical and structural systems*. John Wiley & Sons, New York, N.Y.

Lee, H. (1983). "An application of integer and discrete optimization in engineering design," MS thesis, Univ. of Missouri-Columbia, Columbia, Mo.

Sandgren, E. (1990). "Nonlinear integer and discrete programming in mechanical design." *J. Mech. Des.*, Vol. 112, 223–229.

Siddall, J. N. (1982). *Optimal engineering design: principles and applications*. Marcel Dekker, New York, N.Y.

Thanedar, P. B., Arora, J. S., Tseng, C. H., Lim, O. K., and Park, G. J. (1986). "Performance of some SQP algorithms on structural design problems." *Int. J. Numerical Methods in Engrg.*, Vol. 23, 2187–2203.

Tseng, C. H., and Kao, K. Y. (1989). "Performance of a hybrid sensitivity analysis on structural design problems." *Computers and Struct.*, 33(5), 1125–1131.

Tseng, C. H., and Wang, L. W. (1989). "The application of branch-and-bound method in large number of non-continuous design variables optimization." *Tech. Rep.*, Nat. Chiao Tung Univ., Taiwan, Republic of China.

Vanderplaats, G. N., and Thanedar, P. B. (1991). "A survey of discrete variable optimization for structural design." *Proc., 10th Conf. on Electronic Computation in Struct. Engrg.*, ASCE, New York, N.Y.

APPENDIX II. NOTATION

The following symbols are used in this paper:

Cost_{il} = cost value of lower bounds of ith design variable at node;
Cost_{iu} = cost value of upper bounds of ith design variable at node;
NB = number used in neighboring search;
X_{il} = lower allowable values of ith design variable;
X_{iu} = upper allowable values of ith design variable;
x_{obj} = design variable to be branched;
ΔCost_i = difference between two costs; and
Δx_i = smallest clearance of design variable i.

J. Struct. Eng. 1995.121:831-837.