Management Science

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

Branch and Bound Experiments in Convex Nonlinear Integer Programming

Omprakash K. Gupta, A. Ravindran,

To cite this article:
Omprakash K. Gupta, A. Ravindran, (1985) Branch and Bound Experiments in Convex Nonlinear Integer Programming. Management Science 31(12):1533-1546. http://dx.doi.org/10.1287/mnsc.31.12.1533

MANAGEMENT SCIENCE
Vol. 31, No. 12, December 1985
Printed in U.S.A.

BRANCH AND BOUND EXPERIMENTS IN CONVEX NONLINEAR INTEGER PROGRAMMING*

OMPRAKASH K. GUPTA AND A. RAVINDRAN
Indian Institute of Management, Ahmedabad—380015, *India*
School of Industrial Engineering, University of Oklahoma, Norman, Oklahoma 73019

The branch and bound principle has long been established as an effective computational tool for solving mixed integer linear programming problems. This paper investigates the computational feasibility of branch and bound methods in solving convex nonlinear integer programming problems. The efficiency of a branch and bound method often depends on the rules used for selecting the branching variables and branching nodes. Among others, the concepts of pseudo-costs and estimations are implemented in selecting these parameters. Since the efficiency of the algorithm also depends on how fast an upper bound on the objective minimum is attained, heuristic rules are developed to locate an integer feasible solution to provide an upper bound. The different criteria for selecting branching variables, branching nodes, and heuristics form a total of 27 branch and bound strategies. These strategies are computationally tested and the empirical results are presented.

(PROGRAMMING—INTEGER ALGORITHMS, BRANCH AND BOUND; PROGRAMMING—INTEGER ALGORITHMS, TESTS; PROGRAMMING—NONLINEAR, ALGORITHM TESTS)

1. Introduction

In this paper we consider the solution of nonlinear convex programming problems that require some or all of the decision variables to be integer valued.

In general, a nonlinear mixed integer programming (NLMIP) problem can be stated as follows:

$$\begin{aligned} \text{Minimize} \quad & f(x), && x = (x_1, x_2, \ldots, x_n) \in R^n, \\ \text{Subject to:} \quad & g_i(x) \geqslant 0, && i = 1, 2, \ldots, NI \\ & h_k(x) = 0, && k = 1, 2, \ldots, NE, \\ & x_j \text{ is an integer}, && j = 1, 2, \ldots, m \leqslant n. \end{aligned}$$

It should be noted that out of the n decision variables $x_1, x_2, \ldots, x_n$ the first m variables are identified as integer variables for convenience. Throughout this paper, we will assume that the continuous relaxation of NLMIP (ignoring integer restrictions) satisfies the first order sufficient optimality conditions and hence a Kuhn–Tucker point is a global minimum.

Probably the simplest approach that can be used to solve such problems is that of rounding as in *linear* mixed integer programming. Solve the continuous relaxation of NLMIP and round the continuous optimal solution to the nearest integer point. Though simple in concept, it is well known that this method has many pitfalls (Glover and Sommer 1975). The rounded solution may not only be suboptimal, it may in fact be infeasible! The task of locating a feasible integer solution may in itself be nontrivial in the nonlinear case. However, for problems of special structure, several algorithms have been developed and surveys of such algorithms can be seen in Cooper (1981) and Gupta and Ravindran (1983).

*Accepted by Donald Erlenkotter, former Departmental Editor; received September 9, 1981. This paper has been with the authors 4 months for 1 revision.

0025-1909/85/3112/1533$01.25

2. Branch and Bound Strategies for Nonlinear Integer Programming Problems

The branch and bound method has evolved through the contributions of many researchers, one of the earliest being that of Land and Doig (1960). It consists of a systematic search of continuous solutions in which the integer variables are successively forced to take on integral values; the logical structure of the set of solutions is that of a tree. The algorithm starts by finding a global optimal solution to the continuous problem where the integrality requirements are relaxed. If this solution is integral, then it is an optimal solution to the NLMIP problem. If it is nonintegral then at least one integer variable, say x_j, is continuous. We now divide the value of x_j into integral and fractional parts $[x_j]$ and x_j^* respectively, defined by: $x_j = [x_j] + x_j^*$, where $[x_j]$ is integral and $0 < x_j^* < 1$. We then form two subproblems, one with the additional upper bound constraint $x_j \leqslant [x_j]$, and another with the lower bound constraint $x_j \geqslant [x_j] + 1$. The process of forming these subproblems is called *branching*. Each of these subproblems is solved again as a continuous problem. The information regarding the optimal solution of the continuous problems thus solved is stored in a corresponding *node*. A node generally stores the optimal solution and the corresponding value of the objective function, and also lower and upper bounds on the variables. Note that the objective function values provide a lower bound on the optimal value of the NLMIP problem.

The above procedure of branching and solving a sequence of continuous problems is continued until a feasible integer solution to one of the continuous problems is found. When we obtain one such feasible integer solution, the corresponding value of the objective function becomes an upper bound on the objective of the NLMIP problem. At this point, we can eliminate from further consideration all those continuous solutions or nodes whose values of the objective function are higher than the upper bound and we say that the corresponding nodes are *fathomed*. Nodes are also eliminated, or fathomed, when the continuous problem has a feasible integer solution or is infeasible. This procedure of branching and fathoming is repeated for each of the unfathomed nodes. As observed above, from each node at most two new nodes may originate. Whenever a feasible integer solution is found, and the value of the objective function is found to be less than the upper bound to date, it becomes the new upper bound on the objective. The search for the optimal solution terminates when all the nodes are fathomed. The current best integer solution gives the optimal solution to the given nonlinear integer programming problem.

The general description of branch and bound leaves the following questions unresolved:

(1) What procedure should be used to solve the intermediate nonlinear continuous problems?

(2) What criterion should be used to select the variable to branch upon?

(3) What criterion should be used to select the node for branching?

We will now address each of these questions separately.

2.1. *Selection of Nonlinear Continuous Algorithm*

Since a large number of nonlinear continuous problems need to be solved, it is necessary that we employ an efficient algorithm for their solution. As is widely known, a number of algorithms have been developed for solving nonlinear continuous problems based on a variety of techniques such as penalty function methods (Fiacco and McCormick 1968), Generalized Reduced Gradient (GRG) methods (Abadie and Carpentier 1969, Wolfe 1967), etc. Several computer codes have been developed based on these algorithms. A number of comparative studies (Colville 1968, Eason and Fenton 1974, Sandgren and Ragsdell 1980, Waren and Lasdon 1979) have been

carried out to evaluate the relative performance of these codes. These studies have shown that the codes based on GRG methods are significantly superior to the others. For our experiment, we used the code OPT by Gabriele and Ragsdell (1976) based on GRG. Like most GRG codes, OPT handles problem constraints separately from the variable bounds. This is particularly useful for our application since the branch and bound method simply alters the lower and upper bounds on the variables throughout the solution process.

2.2. *Selection of Branching Variables*

In the case of linear problems it has been known that the rule used for the selection of branching variables may have a significant effect on the overall performance of a branch and bound strategy. The following three variable selection strategies were selected for investigation:

1. *Lowest-Index-First.* It is possible to have some information on the importance of some of the integer variables in a given model. The integer variables are arranged in order of importance, the most important of these being processed first. This is accomplished by indexing the variables with the decreasing priorities of the integer variables and then selecting the variable with the lowest index first.

2. *Most Fractional Integer Variable.* This strategy selects the variable which is farthest from its nearest integer value. This choice is aimed at getting the largest degradation of the objective when branching is carried out so that more nodes can be fathomed at an early stage.

3. *Use of Pseudo-Costs.* The concept of pseudo-cost was first developed by Benichou et al. (1971) for solving mixed integer linear programming problems. The pseudo-costs are used as a quantitative measure of the importance of the integer variables and this allows the assignment of some priority to the variables. For each integer variable x_j two quantities are defined, lower pseudo-cost (pcl_j) and upper pseudo-cost (pcu_j). The values of the lower and upper pseudo-costs are computed during the tree search as follows.

Suppose that at node k the variable x_j is selected for branching. Let x_j^* denote the fractional part of the value of the variable x_j. Let f_k denote the value of the objective at this node k. Let f_L be the value of the objective when the continuous problem is solved with the additional lower bound constraint $x_j \leqslant [x_j]$. The lower pseudo-cost of x_j is defined by: $pcl_j = (f_L - f_k)/x_j^*$.

Let f_U be the value of the objective function when the continuous problem is solved with additional upper bound constraint $x_j \geqslant [x_j] + 1$. The upper pseudo-cost of x_j is defined by: $pcu_j = (f_U - f_k)/(1 - x_j^*)$.

These pseudo-costs may be interpreted as the deterioration of the functional value per unit change of the variable x_j, pcl_j corresponding to the decrease of x_j and pcu_j corresponding to the increase of x_j. Although the values of the pseudo-costs depend on the node where they are computed, they are computed only once and are assumed to remain constant so that the computational effort of recomputing them at every node could be saved. This strategy of selecting the branching variable is invoked in the following manner:

(i) Calculate the lower and upper pseudo-costs for all integer variables when possible.

(ii) Compute the quantity $V_j = \min[pcl_j \cdot x_j^*, pcu_j(1 - x_j^*)]$ for each integer variable x_j.

(iii) Select the variable x_j for which the value of V_j is maximum.

The choice of the branching variable described above is such that the value of the function is expected to increase the most.

2.3. *Selection of Branching Nodes*

Just as a selection variable criterion may affect the performance of a branch and bound strategy, it has been found that the selection method for branching nodes may likewise significantly affect the performance of branch and bound. We implemented the following node selection strategies to see their effect on nonlinear integer programming problems.

1. *Branch from the Node with Lowest Bound.* As the name suggests, in this strategy, the node which currently has the lowest bound on the objective is selected for branching. Lawler and Wood (1966) argued that if for any given problem the set of new bounding problems is uniquely determined, then this strategy has the advantage that the total amount of computation is minimized, in the sense that any branching operation performed under this strategy must also be performed under any other strategy.

2. *Branch from the Newest Node.* In this strategy, whenever a branching is carried out the nodes corresponding to the new problems are given preference over the rest of the unfathomed nodes. The node that is newest in the list of unfathomed nodes is selected for branching. This strategy is known variously as the depth, back-track, or last-in first-out strategy, and has the advantage of saving storage space. As pointed out by Ibaraki (1976), this type of strategy requires an amount of memory that is a linear function of the problem size and is relatively easy to implement.

3. *Use of Estimations.* In the lowest bound selection strategy a node with the lowest bound on the objective is selected. This strategy considers only the value of the objective function and does not take into account the "quality" of the continuous optimal solutions. Let us consider a three-variable problem to illustrate the concept of "quality". Suppose, we need to select a node out of the following unfathomed nodes:

$$x^{(1)} = (1.3, 2.4, 5.1), \quad z_1 = 10.5; \quad \text{and} \quad x^{(2)} = (1.0, 3.0, 5.1), \quad z_2 = 10.6.$$

According to the lowest bound selection scheme, one would prefer node 1 over node 2. However, such a selection does not seem to be quite appropriate as the solution $x^{(2)}$ is much closer to an integer solution. If selected, node 2 is likely to be fathomed in the next iteration whereas node 1 may require several iterations before it can be eliminated. Attempts have been made to modify the lowest bound selection scheme so that the quality of the continuous optimal solutions also is taken into account. In the linear case, this is accomplished by Benichou et al. (1971) using a new concept called *estimation* that uses pseudo-costs.

At node k, a quantity called the estimation of node k, denoted by E_k, is computed by: $E_k = f_k + \sum_{j=1}^{m} V_j$.

The quantity E_k is in some sense an estimate of the value of the objective function of the best integer solution that can be expected for a descendant of node k.

Using the concept of estimations, this node selection strategy is invoked in the following manner:

(i) Compute the estimation of all the unfathomed nodes.

(ii) Select the node with the lowest estimation.

2.4. *Heuristics for Upper Bound*

The lowest value of f_k at any stage provides a lower bound on the objective of the NLMIP problem. Often it is important to find an upper bound on the objective as quickly as possible. This may be accomplished by initially finding an integer feasible solution to the problem. The following two heuristics were developed to find initial integer feasible solutions.

Heuristic A. Let $x = (x_1, x_2, \ldots, x_n)$ be the solution for an unfathomed node and

p represent the number of integer variables assuming nonintegral values in this solution. The number p is defined as the Order of Infeasibility of this node.

Heuristic A is carried out in the following steps:

1. Select a node for which the order of infeasibility is minimum. Select a variable, say x_j, for branching according to a prespecified selection rule. For this node, a number of integer variables already may have integer values. These variables are fixed to their corresponding integer values while branching. Thus, they are not allowed to change their values in either of the two subproblems. The idea is to decrease the number of integer variables with non-integer values, and thus work toward solutions with lower orders of infeasibility.

2. If the subproblem with upper bound constraint $x_j \leqslant [x_j]$ is "continuous feasible," fix the variable x_j to $[x_j]$ in all the subsequent descendants of this node. Similarly, if the subproblem with the lower bound constraint $x_j \geqslant [x_j] + 1$ is "continuous feasible," fix the variable x_j to $[x_j] + 1$ in all the subsequent descendants of this node.

If after solving the two subproblems an integer feasible solution is obtained, stop. Otherwise, return to step 1.

Heuristic B. For the sake of notational simplicity, assume that all the variables are integer variables. Heuristic B is carried out as follows:

1. Solve the nonlinear continuous problem. Let $x = (x_1^0, x_2^0, \ldots, x_n^0)$ be the continuous optimal solution. Find the integer y_j nearest to x_j^0, for $i = 1, 2, \ldots, n$; and form a vector $y = (y_1, y_2, \ldots, y_n)$. Initialize a counter $i = 0$.

2. If y is a continuous feasible solution, go to step 4. Otherwise, go to step 3.

3. Reset the counter i to $i + 1$, and replace the value of the variable x_i by x_i^0. Also replace vector y by $y = (x_1^0, x_2^0, \ldots, x_i^0, y_{i+1}, \ldots, y_n)$, and return to step 2.

4. If y is integer feasible, stop. Otherwise, go to step 5.

5. Construct an i-dimensional problem by substituting the variables $x_{i+1}, x_{i+2}, \ldots, x_n$ by the integer values $y_{i+1}, y_{i+2}, \ldots, y_n$ in the objective and the constraint functions of the given problem. Thus an i-dimensional nonlinear integer programming problem is constructed with $x_1, x_2, \ldots, x_i$ as its integer variables. Starting with $(x_1^0, x_2^0, \ldots, x_i^0)$ as an initial solution, apply branch and bound to solve this i-dimensional problem. If any of the descendants of this problem yields an integer feasible solution, say $(z_1, z_2, \ldots, z_i)$, stop. The vector $x = (z_1, z_2, \ldots, z_i, y_{i+1}, \ldots, y_n)$ would be an integer solution to our original problem. Otherwise, return to step 3.

It should be noted that in the beginning of the heuristic the problems are smaller in size, and hence may be solved quickly. However, as the solution process develops, the problems become larger in size, and in the worst possible situation the original problem itself would be solved. If the original problem has an integer feasible solution, such a solution would be discovered in our search. Therefore, this heuristic guarantees that an integer feasible solution will be obtained whenever one exists for the given problem.

3. Experimental Design

The primary variables evaluated in the experimental work were the strategies for branching variable selection and node selection. Three options for each of these two selection categories provided nine strategy combinations. For each combination, we explored three options on the heuristics: (1) no heuristic, (2) Heuristic A, and (3) Heuristic B. Thus we had a total of 27 branch and bound strategies comprising of an experimental design with three levels of treatment. A computer code BBNLMIP developed by Gupta (1980) that was capable of invoking each of the 27 strategies was used in the study. BBNLMIP employs the nonlinear code OPT developed by Gabriele

and Ragsdell (1976) to solve the nonlinear continuous problems. Each strategy was identified by the parameters (K, L, N) where K was the option for the branching variable selection, L for the branching node selection, and N for the heuristic selection. The options for K, L and N were denoted as follows:

$K = 1$: Lowest index first.
$K = 2$: Most fractional integer variable.
$K = 3$: Use of pseudo costs.
$L = 1$: Branch from the node with the lowest objective value.
$L = 2$: Last-in first-out.
$L = 3$: Branch from the node with the lowest estimation.
$K = 1$: No heuristic.
$K = 2$: Select Heuristic A.
$K = 3$: Select Heuristic B.

3.1. *Test Problems*

In any computational study, it is important to examine a wide range of test problems. It should be pointed out that in contrast to its linear counterpart, in nonlinear integer programming, the problem size is only one of many factors in determining its overall complexity; the degree of nonlinearity of the objective and the constraint functions including the number of nonlinear terms contributes significantly to the complexity of the problem.

Since, to our knowledge, no experimental study of this nature has been carried out, we considered, among others, test problems from the various past studies on nonlinear continuous algorithms such as Colville (1968), Eason and Fenton (1974), Himmelblau (1972), and Sandgren and Ragsdell (1980). Among the available test problems, we considered only those convex problems where certain variables can be treated as integer variables; e.g., those problems having some of the variables with reasonably tight bounds and feasible regions with integer points. It serves no purpose to consider those problems that have no integer points in their feasible region or where the variables take on large values. A total of 14 real-world problems were selected from these studies as well as other literature sources. We also generated a set of seven additional convex problems using the technique of Rosen and Suzuki (1965). This technique generates nonlinear convex problems with quadratic objective and quadratic constraints. It assumes a known primal and dual solution and generates randomly an appropriate Kuhn–Tucker problem. One problem was constructed by the authors during the development and initial testing of the computer code BBNLIP. This was also included as a test problem. Thus, we had a total collection of 22 test problems. The test problems ranged from two to sixteen continuous variables, and two to ten integer variables. Out of the 22 problems, problems 5, 8, 20 and 21 were of the mixed nonlinear integer programming type, and the rest were pure nonlinear integer programming problems. The number of constraints varied from zero to eight. The largest problem that was solved was problem 20 with 24 variables (8 integer) and 8 linear constraints involving 16 terms in each constraint. The objective function was a fourth degree polynomial involving 45 nonlinear terms.

As Table 1 illustrates, problem size alone does not determine the computational effort. Problem 19 which took the largest time to solve had only 16 variables (8 integer) and 8 constraints. However, its objective function involved 40 quadratic terms and each of the constraints had between 25 and 32 quadratic terms. Similarly, problem 17 had only 14 variables (7 integer) and 7 constraints, but the objective had 31 quadratic terms and each constraint had between 24 and 28 quadratic terms. The objective functions of all the problems were nonlinear; however problems 4, 6, 9 and 16 had no constraints and problems 3, 15, and 20 had linear constraints. Additional problems of

TABLE 1

Actual Solution Times (seconds)

Problem	Average Solution Time	Worst Solution Time	Best Solution Time	Ratio Worst/Best
1	1.035	1.433	0.822	1.74
2	12.038	22.150	5.339	4.15
3	0.759	0.875	0.704	1.24
4	0.466	0.527	0.428	1.23
5	6.250	6.835	5.720	1.19
6	0.563	0.651	0.521	1.25
7	0.856	1.136	0.631	1.80
8	2.504	2.991	2.136	1.40
9	3.256	4.707	2.527	1.86
10	0.728	0.921	0.626	1.47
11	2.098	2.902	1.401	2.07
12	4.816	8.044	2.760	2.91
13	13.668	24.677	7.660	3.22
14	12.196	20.748	9.679	2.14
15	1.303	1.581	1.131	1.40
16	0.624	0.828	0.503	1.65
17	108.131	220.418	49.378	4.46
18	83.554	237.756	33.377	7.12
19	241.522	496.349	90.992	5.45
20	66.261	109.011	50.257	2.17
21	1.945	2.061	1.877	1.10
22	8.233	12.118	5.973	2.03
			Average Ratio	2.41

greater difficulty were also considered, but the excessive computational cost made it impractical for us to use them as test problems. A complete listing of test problems can be found in Gupta (1980).

4. Results and Analysis

Each of the 27 strategies was tested on all 22 test problems on the CDC 6500 Computer system at Purdue University. Except for Problem 19, every strategy was able to solve each problem in less than 240 seconds of execution time. Problem 19 turned out to be more difficult to solve, as two of the strategies, $(K, L, N) = (1, 2, 1)$ and $(1, 2, 3)$ could not find optimal solutions within a time limit of 500 seconds. Two other strategies, $(3, 2, 3)$ and $(3, 3, 3)$ generated subproblems for Problem 19 for which the code OPT failed to find even continuous optimal solutions. Although the remaining 23 strategies did find optimal solutions to this problem, some strategies took almost 500 seconds to reach the final solution. The fact that four strategies were unable to reach the final solution to one of the 22 test problems detracts little from the performance of these strategies or the code OPT itself since they were successful on all the other problems.

4.1. *A Strategy Ranking Criterion*

The significant cost for executing a nonlinear integer program is the computer solution time. Therefore we decided to use solution time as the initial indicator of the performance of the strategies. It was found that our test problems had a wide variation in solution times, ranging from less than 0.5 second to almost 500 seconds. Table 1 presents the average solution times, the worst solution times and the best solution times. It also gives the ratio of the worst to the best solution times.

Since our primary interest is to compare various branching variables, node, and heuristic selection strategies, we first computed the average solution times for each test

TABLE 2
Average Solution Times for Branching Variable Options

Problem	$k = 1$	$k = 2$	$k = 3$
1	1.323	0.891	0.891
2	13.918	10.224	11.972
3	0.760	0.760	0.759
4	0.466	0.467	0.466
5	6.257	6.241	6.251
6	0.564	0.563	0.561
7	1.053	0.755	0.759
8	2.702	2.410	2.400
9	3.265	3.240	3.262
10	0.726	0.730	7.282
11	2.134	2.078	2.081
12	4.867	4.794	4.788
13	13.099	13.880	14.034
14	11.507	11.766	13.316
15	1.168	1.372	1.367
16	0.754	0.557	0.561
17	124.705	92.828	107.971
18	78.826	44.470	124.033
19	271.261	157.641	262.163
20	53.290	70.459	74.403
21	1.945	1.949	1.940
22	6.720	6.768	11.210
Overall Average	27.361	19.766	29.658

problem taken over the nine combinations while keeping the value of parameters fixed one at a time. The results are shown in Tables 2, 3, and 4. As shown in Table 2, the overall average of the solution times by the branching variable selection option 2 (i.e., selecting the most fractional integer variable) is 19.766 seconds which is significantly less than the overall averages obtained by the other two variable selection options. Similarly Table 3 shows that the node selection option 2 is significantly worse than the other two as its overall average solution time is relatively very large. Table 4 indicates that the heuristics do not particularly provide any advantage in reducing the computational time. We have made this analysis by considering only the actual solution times. A closer look at solution times however indicates that a bulk of the time is spent on Problems 17, 18, 19 and 20. And, therefore, a simple overall average of the solution times could be misleading due to differential weighting. Hence, we decided to make another comparison using "normalized solution times" as opposed to actual times.

Normalized Solution Time (*NST*). To give equal importance to each test problem in our study, the following procedure was used to normalize the solution times.

1. Let t_{ij} denote the solution time for the ith strategy to solve the jth problem, and n_j denote the number of strategies that were able to reach the final optimal solution for the jth problem within a specified time limit of 500 seconds ($i = 1, 2, \ldots, 27$; $j = 1, 2, \ldots, 22$).

2. Form the sum $S_j = \sum_i t_{ij}$ where the sum is taken over all those strategies i which could solve the problem j; thus, the average solution time for the jth problem $A_j = S_j/n_j$ ($j = 1, 2, \ldots, 22$).

3. Form the normalized solution times (NST_{ij}) for the ith strategy to solve the jth problem as: $NST_{ij} = t_{ij}/A_j$.

Since the average normalized solution time for each problem is one unit, the test problem results are now comparable, and we can make direct comparisons. A strategy with lower (less than one) normalized solution times would have a better rating than a strategy with relatively higher (greater than one) normalized solution times. The

TABLE 3

Average Solution Time for Node Selection Options

Problem	$L = 1$	$L = 2$	$L = 3$
1	1.036	1.038	1.032
2	14.396	11.157	10.562
3	0.760	0.758	0.761
4	0.469	0.466	0.464
5	6.256	6.242	6.251
6	0.562	0.565	0.561
7	0.852	0.851	0.864
8	2.420	2.688	2.405
9	3.254	3.254	3.260
10	0.676	0.840	0.668
11	1.785	2.721	1.788
12	3.920	6.579	3.950
13	10.171	20.910	9.922
14	10.277	16.110	10.253
15	1.297	1.346	1.277
16	0.611	0.651	0.610
17	83.139	157.788	83.467
18	76.187	105.564	68.912
19	162.101	398.140	178.578
20	62.455	73.898	62.429
21	1.950	1.936	1.949
22	7.968	8.671	8.059
Overall Average	20.568	36.462	20.819

TABLE 4

Average Solution Times for Heuristic Selection Options

Problem	$N = 1$	$N = 2$	$N = 3$
1	0.981	1.145	0.980
2	10.093	16.414	9.607
3	8.711	0.854	0.713
4	0.441	0.518	0.440
5	5.771	6.176	6.802
6	0.534	0.622	0.530
7	0.737	0.916	0.915
8	2.383	2.731	2.399
9	2.554	4.658	2.555
10	0.690	0.798	0.696
11	1.785	2.266	2.243
12	4.355	5.985	4.110
13	12.394	16.190	12.418
14	12.000	12.550	12.039
15	1.304	1.353	1.252
16	0.624	0.665	0.583
17	103.813	123.919	96.661
18	79.389	85.895	85.379
19	189.179	221.563	242.506
20	57.682	83.749	57.351
21	1.906	2.021	1.908
22	7.959	8.816	7.922
Overall Average	22.604	27.264	25.000

TABLE 5
Ranking of Strategies

Rank	K	L	N	Average NST
1	2	3	1	0.779
2	2	1	1	0.790
3	2	3	3	0.816
4	2	1	3	0.839
5	1	3	1	0.867
6	3	3	1	0.869
7	1	1	1	0.877
8	3	1	1	0.897
9	1	3	3	0.909
10	3	3	3	0.917
11	1	1	3	0.932
12	3	1	3	0.989
13	2	3	2	0.994
14	2	2	3	1.009
15	2	1	2	1.026
16	3	3	2	1.031
17	2	2	1	1.037
18	3	2	3	1.093
19	1	1	2	1.099
20	1	2	3	1.108
21	3	2	1	1.109
22	3	1	2	1.113
23	1	3	2	1.133
24	1	2	1	1.141
25	2	2	2	1.193
26	1	2	2	1.257
27	3	2	2	1.373

following procedure is used to evaluate the performance of the strategies:

1. First compute the normalized solution times (NST_{ij}) for each strategy and test problem.

2. Form the sum $ST_i = \sum_j NST_{ij}$, and the average $AV_i = ST_i/N_i$, where N_i is the number of problems solved by strategy i.

3. Rank the average normalized solution times for each strategy, the best strategy being the one with the lowest value of AV_i.

The ranking and average normalized solution times are shown in Table 5. The following observations can be made:

1. None of the top 12 strategies invokes either branching from the newest node or implementation of Heuristic A. In fact, these 12 strategies comprise the exclusive and exhaustive combination of the other selection strategies (a $3 \times 2 \times 2$ design).

2. The top four strategies invoke the most fractional integer variable as the branching variable selection strategy, and the next eight strategies use the other two branching variable selection strategies. The pattern of the most fractional integer variable selection strategy being better than the other two strategies continues when a combination of node selection and heuristic selection is fixed, and therefore we may conclude that most fractional integer variable strategy is a relatively better strategy than the other two strategies for the branching variable selection in these problems.

Analysis of Variance. In order to make statistical comparisons, the statistical package SPSS was used to analyze the data provided by the normalized solution times for the purpose of the analysis of variance. Before considering the table of analysis of variance, it is of interest to examine the overall mean values of the three selection parameters. Tables 6, 7, and 8 present the overall means for the branching variable selection, node selection and heuristic selection strategies. The results are clearly

TABLE 6

Mean Values of NST for Branching Variable Selection Strategies

Lowest Index First	1.03
Most Fractional Variable	0.94
Pseudo Costs	1.04

TABLE 7

Mean Values of NST for Node Selection Strategies

Lowest Bound	0.95
Newest Node	1.15
Estimation	0.92

TABLE 8

Mean Values of NST for Heuristic Selection Strategies

No Heuristic	0.93
Heuristic A	1.14
Heuristic B	0.96

TABLE 9

Analysis of Variance (Normalized Solution Time)

Source of Variation	Sum of Squares	D.F.	Mean Square	*F*-Ratio	Significance
Main Effects	12.000	6	2.000	30.621	0.001
Variable Selection	1.243	2	0.621	9.513	0.001
Node Selection	5.783	2	2.892	44.272	0.001
Heuristic Selection	4.937	2	2.469	37.796	0.001
2-Way Interaction	0.229	12	0.019	0.293	0.991
Variable—Node	0.076	4	0.019	0.292	0.883
Variable—Heuristic	0.010	4	0.002	0.037	0.997
Node—Heuristic	0.142	4	0.036	0.545	0.703
3-Way Interactions	0.251	8	0.031	0.480	0.870
Variable-Node-Heuristic	0.251	8	0.031	0.480	0.870
Explained	12.480	26	0.480	7.349	0.001
Residual	36.773	563	0.065		
Total	49.253	589	0.084		

consistent with the observations made earlier. The most fractional integer variable selection strategy performs the best among branching variable selection strategies. The other two branching variable selection strategies perform about the same. Selecting from the newest node turns out to be the worst of the node selection strategies. The strategy using estimations seems to perform better than the one that selects the node with the lowest bound. None of the heuristics seems to provide any computational advantage.

With these broad observations in mind, the Analysis of Variance can be examined in Table 9. The results shown there indicate that significant differences exist among the

TABLE 10
Ranking of Strategies

Rank	K	L	N	Average NLP
1	2	3	1	0.781
2	2	1	1	0.802
3	3	3	1	0.827
4	1	3	1	0.836
5	1	1	1	0.839
6	2	3	3	0.840
7	3	3	3	0.851
8	3	1	1	0.859
9	2	1	3	0.862
10	1	1	3	0.899
11	1	3	3	0.902
12	3	1	3	0.906
13	2	3	2	1.013
14	2	1	2	1.040
15	3	3	2	1.044
16	1	1	2	1.075
17	1	3	2	1.082
18	3	1	2	1.096
19	2	2	3	1.101
20	2	2	1	1.108
21	3	2	3	1.131
22	1	2	3	1.143
23	3	2	1	1.190
24	1	2	1	1.200
25	1	2	2	1.218
26	2	2	2	1.249
27	3	2	2	1.302

means of solution times for all the three main effects of selection strategies. This is not surprising given the preceding discussion. The table also indicates that there are no significant differences due to two-way interactions. This provides evidence that the three selection parameters are independent of one another.

4.2. *Another Strategy Ranking Criterion*

In the above ranking procedure, we have compared the normalized solution times which are computed from their corresponding actual solution times. The solution time often depends on the following:

1. The particular computer system used for the study.

2. The particular computer program used to solve the problem. In our case, the computer node BBNLMIP involves the following:

a. The branch and bound computer code where all the logical operations of the branch and bound are performed except for solving the nonlinear continuous problems, and

b. The nonlinear code OPT which solves all the intermediate continuous problems.

Since the computational time depends on the computer system, the branch and bound code, and the nonlinear code (OPT, in our case), the ranking of the strategies might as well depend on these factors. Hence, another ranking criterion is also used to make comparisons where these factors would not have any effect.

This criterion compares the *number of nonlinear continuous problems* solved instead of the solution times. The continuous problems that are solved under a particular branch and bound strategy in fact define the corresponding branch and bound tree and therefore these problems would remain the same irrespective of the computer system, the branch and bound code, and the nonlinear algorithm used to solve the problem.

TABLE 11

Mean Values of NNLP for Branching Variable Selection Strategies

Lowest Index First	1.02
Most Fractional Variable	0.98
Pseudo-Costs	1.02

TABLE 12

Mean Values of NNLP for Node Selection Strategies

Lowest Bound	0.93
Newest Node	1.18
Estimation	0.91

TABLE 13

Mean Values of NNLP for Heuristic Selection Strategies

No Heuristic	0.94
Heuristic A	1.12
Heuristic B	0.96

The total number of nonlinear continuous problems solved by each strategy was tabulated and a procedure similar to one used for normalizing the solution times was used to normalize the number of nonlinear continuous problems solved for each problem. The strategies were ranked according to their average value of normalized number of nonlinear problems (NNLP). The results are displayed in Table 10.

Observe that the top 12 strategies still remain the same. None of them makes use either of branching from the newest node or of implementing Heuristic A. The top two strategies are still the same. The rankings of the other strategies do not change significantly. The overall means for the selection parameters are given in Tables 11, 12, and 13. The results are similar to those of Tables 6, 7, and 8.

5. Conclusions

In this paper, we have investigated the feasibility of applying the branch-and-bound approach to nonlinear convex integer programming problems. As we have shown, branch-and-bound methods can be implemented as useful computational tools for solving such problems. We have tested the concepts of pseudo-costs and estimations in selecting branching variables and branching nodes. The results show that among the three branching variables selection options we have tested, the strategy of selecting the integer variable with the most fractional value was the best. Among node selection strategies, the criterion of selecting the newest node was the worst. It should be noted that considerable effort is made in computing pseudo-costs and estimations at the beginning of each run, and therefore their full strength probably is not exploited in less complex test problems. We would like to point out that even though the branch and bound method is not guaranteed to solve nonconvex problems, we have successfully solved several nonconvex problems using the BBNLIP code.

References

ABADIE, J. AND J. CARPENTIER, "Generalization of the Wolfe Reduced Gradient Method to the Case of Nonlinear Constraints," *Optimization*, R. Fletcher (Ed.), Academic Press, New York, 1969, 37–47.

BENICHOU ET AL., "Experiments in Mixed-Integer Linear Programming," *Math. Programming*, 1, 1 (1971), 76–94.

COLVILLE, A. R., "A Comparative Study of Nonlinear Programming Codes," Technical Report No. 320-2949, IBM New York Scientific Center, June 1968.

COOPER, MARY W., "A Survey of Methods for Pure Nonlinear Integer Programming," *Management Sci.*, 27, 3 (1981), 353–361.

EASON, E. D. AND R. G. FENTON, "A Comparison of Numerical Optimization Methods for Engineering Design," *ASME J. Engineering for Industry*, 96, 1 (1974), 196–200.

FIACCO, A. V. AND G. P. MCCORMICK, *Nonlinear Programming: Sequential Unconstrained Minimization Techniques*, John Wiley, New York, 1968.

GABRIELE, G. A. AND K. M. RAGSDELL, "OPT, A Nonlinear Programming Code in FORTRAN IV," The Modern Design Series, Vol. 1, Purdue Research Foundation, 1976.

GLOVER, F. AND D. SOMMER, "Pitfalls of Rounding in Discrete Management Decision Variables," *Decision Sci.*, 22, 4 (December 1975), 455–460.

GUPTA, OMPRAKASH K., "Branch and Bound Experiments in Nonlinear Integer Programming," Ph.D. Thesis, School of Industrial Engineering, Purdue University, West Lafayette, Ind., December 1980.

——— AND A. RAVINDRAN, "Nonlinear Integer Programming Algorithms: A Survey," *Opsearch*, 20 (December 1983), 189–206.

HIMMELBLAU, D. M., *Applied Nonlinear Programming*, McGraw-Hill, New York, 1972.

IBARAKI, T., "Theoretical Comparison of Search Strategies in Branch and Bound Algorithms," *Internat. J. Computer and Information Sci.*, 5, 4 (1976), 315–344.

LAND, A. H. AND A. G. DOIG, "An Automatic Method of Solving Discrete Programming Problems," *Econometrics*, 28 (1960), 497–520.

LAWLER, E. L. AND D. E. WOOD, "Branching and Bound Methods: A Survey," *Oper. Res.*, 14 (1966), 699–719.

ROSEN, J. B. AND S. SUZUKI, "Construction of Nonlinear Programming Test Problems," *Comm. ACM*, 8, 2 (1965), 113.

SANDGREN, E. AND K. M. RAGSDELL, "The Utility of Nonlinear Programming Algorithms: A Comparative Study. Parts I and II," *ASME J. Mechanical Design*, 102, 3 (1980), 540–551.

WAREN, A. AND L. LASDON, "The Status of Nonlinear Programming Software," *Oper. Res.*, 27, 3 (1979), 431–456.

WOLFE, P., "Methods for Linear Constraints," *Nonlinear Programming*, J. Abadie (Ed.), North-Holland, Amsterdam, 1967, 99–131.