INTERNAL REPORTS IN

SIMULATION, OPTIMIZATION AND CONTROL

No. SOC-4

A GENERAL PROGRAM FOR DISCRETE OPTIMIZATION

J.H.K. Chen and J.W. Bandler

June 1973

FACULTY OF ENGINEERING
McMASTER UNIVERSITY
HAMILTON, ONTARIO, CANADA

A General Program For Discrete Optimization

J.H.K. Chen and J.W. Bandler
Department of Electrical Engineering
McMaster University
Hamilton, Ontario, Canada

Abstract A general user-oriented computer program for nonlinear discrete optimization problems is presented. The program is based on Dakin's tree search algorithm for mixed integer programming problems. The constrained problem at each stage of the search is transformed into an unconstrained objective by the Bandler-Charalambous technique. The resulting nonlinear programming problem is then solved by the Fletcher-Powell or Fletcher gradient minimization algorithms.

INTRODUCTION

The nonlinear programming problem may be stated as follows. Minimize $f(\underset{\sim}{x})$ subject to

$$g_i(\underset{\sim}{x}) \geq 0, \quad i = 1, 2, \ldots, m$$

where $f(\underset{\sim}{x})$ is the objective function, the vector $\underset{\sim}{x}$ represents a set of k parameters,

$$\underset{\sim}{x} \triangleq \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \end{bmatrix}$$

This work was supported by the National Research Council of Canada under grants A7239 and C154.

and $g_1(\underset{\sim}{x})$, $g_2(\underset{\sim}{x})$, ..., $g_m(\underset{\sim}{x})$ are the inequality constraints. Both f and the g_i's are, in general, nonlinear functions of the parameters. Since minimizing a function is the same as maximizing the negative of the function, there is no loss of generality in considering only minimization.

When all the variables in an otherwise nonlinear programming problem are further restricted to take on integer values only, we have a nonlinear integer programming problem. When only some but not all variables are further constrained to integer values, we have a mixed integer programming problem. More generally, when some of the variables are required to have certain discrete values, we have a discrete value programming problem.

In practical circuit design problems, a compromise between maximum performance and minimum cost is often necessary because usually only components of certain discrete values are available on the market. Components of other values have to be custom-made and are therefore costly. This is but one of the many possible applications of discrete optimization.

INTEGER PROGRAMMING

One approach to integer programming would be to evaluate the function at all integer combinations $\underset{\sim}{x}$ satisfying the given constraints. In most practical cases, because of the vast number of such possible combinations,

this technique would be computationally disastrous. Random search techniques may greatly reduce the amount of computation required but there is no guarantee that the optimum integer solution would be obtained.

Although truncating the fractional part or rounding off of a continuous solution to the nearest set of feasible integers may sometimes yield an excellent approximation of the optimum integer solution when the solution values are large numbers, it does not, in general, provide a solution to the integer programming problem either. Figure 1 illustrates why this is so. By rounding off or truncating the continuous solution 0, we would arrive at one of the four possible integer solutions A, B, C or D. However, it is obvious that the optimum integer solution is point I.

The need for a systematic procedure which will identify the optimum integer solution is thus apparent. In 1958, R.E. Gomory [1] developed the cutting plane algorithms for both the all-integer and the mixed linear programming problems. Another way of solving integer linear programming problems is to reformulate them as 0, 1 problems [2] and then solve via implicit enumeration. In an n-variable 0, 1 problem there will be 2^n possible solutions.

In 1960, Land and Doig [1] proposed another algorithm for solving linear all-integer or mixed programming problems. the procedure consists of a systematic search of continuous

solutions in which integer variables are successively forced to take integer values. The successive iterations are made in an orderly manner so that it is never possible to by-pass a superior integer point in the solution space. The optimum solution is reached at the kth iteration if, for the first time, all the integer variables assume integer values. As it stands the algorithm gives rise to substantial practical difficulties for computer implementation becuase of large storage requirements. In 1966, R.J. Dakin [3] presented a modification of the same algorithm which makes it more amenable to computer coding. The new algorithm is applicable to both linear and nonlinear programming problems. The computer program presented in this report is based on this latter algorithm and it will thus be considered in greater detail in the next section.

DAKIN'S TREE-SEARCH ALGORITHM FOR MIXED INTEGER PROGRAMMING PROBLEMS

The procedure falls into the general classification of branch and bound techniques, as described by Mitten [4] or Lawler and Wood [5]. The algorithm first finds a solution to the continuous problem. If this solution happens to be integral, the integer problem is solved. If it is not, then at least one of the integer variables, e.g.,

x_i, is non-integral and assumes a value x_i^*, say, in this solution. The range

$$[x_i^*] < x_i < [x_i^*] + 1$$

where $[x_i^*]$ is the largest integer value included in x_i^*, is inadmissible and consequently we may divide all solutions to the given problem into two non-overlapping groups, namely,

(1) solutions in which

$$x_i \geq [x_i^*] + 1$$

(2) solutions in which

$$x_i \leq [x_i^*]$$

Each of the constraints is added to the continuous problem and the corresponding augmented problems are solved. The procedure is repeated for each of the two solutions so obtained. Each node represents a solution to a continuous problem and only two branches may emanate from a node. The tree will always terminate either with an integer solution or with a non-feasible solution which violates the current set of constraints. The best solution is then selected from among all the feasible solutions.

It may not be necessary, however, to exhaust all the branches of the tree before the optimum integer solution is attained. Once a feasible integer solution is obtained for one of the branches, its corresponding objective function

value is used as an upper bound so that any new branch yielding a larger objective function value would be terminated. Further addition of parameter constraints would only increase the function value. The upper bound has to be updated, of course, if a better integer solution is encountered.

It seems, then, that the most efficient way of searching would be to branch, at each stage, from the node with the lowest objective function value. This would minimize the searching of unlikely subtrees. To do this, all information about a node has to be retained for comparison and this may require excessive storage for computer implementation. One way of compromising is to search the tree in an orderly manner; each branch is followed until an infeasible or integer solution is reached, or until the objective function value equals or exceeds that for the current best integral solution. All pertinent information regarding the current status of the search is reduced to a list. Each list entry corresponds to a node and at any stage the list represents the chains of nodes leading from the continuous solution node to the current node. A list entry is marked when both branches emanating from a node have been explored. Other information in a list entry includes the name of the specific integer variable considered, the value

of the upper bound and the position of the node. Figure 2 shows the flow diagram for the computational procedure.

The tree is not, in general, unique for a given problem. The tree structure depends on the integer variable, x_i, used to form the parameter constraint at any stage. The amount of computation may be vastly different for different trees.

DISCRETE OPTIMIZATION

For discrete optimization, the algorithm is modified as follows. Let q be the quantization step and x_i be the discrete value variable which assumes a non-discrete solution x_i^*, then the two parameter constraints added alternatively at each node become

(1) $$x_i \geq [x_i^*/q]\, q + q$$

(2) $$x_i \leq [x_i^*/q]\, q$$

The integer problem is thus a special case of discrete optimization with q = 1.

To solve the nonlinear programming problems at the beginning and at each subsequent stages, the Fletcher-Powell [6] or the Fletcher [7] gradient minimization algorithms are employed. The constraints in the problems are taken care of by the Bandler-Charalambous transformation technique [8].

THE COMPUTER PROGRAM

The discrete optimization program is written in the FORTRAN IV language. A number of subprograms were taken from OPTISEP [9] and CONOPT [10] and modified as necessary to fit in with the present package. Figure 3 displays the overall structure of the package. A list with brief description of all the subprograms is given below.

ADDPC	Supplies additional parameter constraints for discrete optimization
ORGNLP	Defines the objective function and the constraints
GRADPC	Returns the gradients of the additional parameter constraints
MINMAX	Transforms the constrained problem into an unconstrained objective by the Bandler-Charalamobus method
FUNCT	Defines the unconstrained objective function and the gradients of the objective function and the constraints
VMMO1	Minimizes a function using the Fletcher optimization algorithm
FMFP	Minimizes a function using the Fletcher-Powell method

DATA	Reads the input data
INPUT	Prints the input data
WRITE1 and WRITE2	Print the intermediate results
FINAL	Outputs the optimum solution after function minimization at each stage
OUTPUT	Outputs the optimum discrete solution in a standard form
GRNLP	Optimizes the original and augmented nonlinear programming problems by one of the two above mentioned gradient minimization methods
DISOPT	Solves the discrete optimization problem based on Dakin's tree-search algorithm. The problem must be formulated in such a way that the discrete variables are the first k design variables of the problem where k is the number of discrete variables. The optimum solution does not have exact discrete values. The program treats any value, which does not differ from the discrete value by more than a prespecified amount, as the discrete value.

A user of the package has to write a small main program to define the input parameters and provide proper dimensioning for the various working arrays and matrices. He also has to write the service subroutines ORGNLP and FUNCT to define the objective function, the original constraints and the gradients of his problem. Other subroutines are general subprograms and may be stored on permanent file. Detailed instructions for programming as well as the listing of the package are given in the appendix.

EXAMPLES

Five minimization problems have been included here to demonstrate the use of the program. The problems were run on a CDC 6400 computer.

Example 1

$$\text{Minimize } f(\underset{\sim}{x}) = x_1^2 + 4x_2^2$$

subject to

$$x_1 + 2x_2 - 1.2 = 0$$

$$x_1,\ x_2 = k \text{ to within } 0.001$$

where k is an integer.

There are two optimum integer solutions $\overset{\vee}{\underset{\sim}{x}} = [2\ 0]^T$ or $\overset{\vee}{\underset{\sim}{x}} = [0\ 1]^T$ with $\overset{\vee}{f} = 4$ as shown in Figures 4, 5, 6 and 7. A much simpler tree is obtained by forcing, at each stage, the variable x_2 to integer value first. This illustrates that the choice of a different integer variable to form the next parameter constraint can have a large effect on the amount of computation required. The Fletcher method was found to be more efficient than the Fletcher-Powell method; the former method required less function evaluations.

If, instead of being integers, the variables x_1 and x_2 are subject to the constraint

$$x_1,\ x_2 = 0.5k \text{ to within } 0.001$$

where k is an integer, then the optimum discrete solution is given by $\overset{\vee}{\underset{\sim}{x}} = [0.5\ 0.5]^T$ and $\overset{\vee}{f} = 1.2500$. The corresponding tree is shown in Figure 8.

In this example, the optimum discrete solution can actually be obtained by rounding off or truncating the continuous solution to the nearest set of discrete solutions in the feasible region and comparing the corresponding function values. To show that this technique is not always applicable, a slightly different problem will be considered next.

Example 2

$$\text{Minimize } f(\underset{\sim}{x}) = x_1^2 + 6x_2^2$$

subject to

$$x_1 + 2x_2 - 1.2 \geq 0$$

$$x_1,\ x_2 = k \text{ to within } 0.001$$

where k is an integer.

The optimum integer solution is $\overset{\vee}{\underset{\sim}{x}} = [2\ 0]^T$ and $\overset{\vee}{f} = 4$. The contour plot and the tree structure are shown in Figures 9 and 10, respectively. In this case, the optimum integer solution is not obtainable by rounding off or truncating the continuous solution $\underset{\sim}{x} = [0.7200\ 0.2400]^T$.

Example 3

$$\text{Minimize } f(\underset{\sim}{x}) = \sum_{i=1}^{3} (C_i - x_1(1 - x_2)^i)^2$$

where $C_1 = 1.5$, $C_2 = 2.25$ and $C_3 = 2.625$,

subject to

$$x_1 \leq 5$$

$$x_1,\ x_2 = k \text{ to within } 0.001$$

where k is an integer.

The objective function has a narrow curving valley

approaching $x_2 = 1$. The contour plot and the results of the problem are shown in Figures 11 and 12, respectively. The optimum integer solution is $\overset{v}{\underset{\sim}{x}} = [2\ 0]^T$ with $\overset{v}{f} = 0.7037$.

Example 4

$$\text{Minimize } f(\underset{\sim}{x}) = 9 - 8x_1 - 6x_2 - 4x_3 + 2x_1^2 + 2x_2^2 + x_3^2 + 2x_1x_2 + 2x_1x_3$$

subject to

$$x_1 + x_2 + 2x_3 \leq 3$$

and, in addition, x_1, x_2 and x_3 are nonnegative integers. For this problem, there are altogether 13 feasible integer solutions as shown in Table I.

All the three optimum integer solutions of unity function value are detected by the algorithm. However, exhaustive enumeration may seem more attractive for this problem because of the small number of possible integer combinations. In a larger problem, one involving more variables and less restrictive constraints, or if the quantization step is reduced, exhaustive enumeration becomes impractical and the efficiency of the algorithm would be more apparent.

The tree structure for Example 4 is shown in Figure 13.

Example 5

Minimize $f = 1/e_1 + 1/e_2$

with respect to e_1, e_2, x_1^o, x_2^o subject to $R_t \subseteq R_c$

where

$$R_t = (\underset{\sim}{x} \mid x_i^o - e_i \leq x_i \leq x_i^o + e_i,\ i = 1, 2)$$

$$R_c = (\underset{\sim}{x} \mid x_1^2 + x_2^2 \leq 4,\ x_1 \geq 0.5,\ x_2 \geq 0.5)$$

and

e_1, e_2 = 0.1k to within 0.001

where k is a nonnegative integer.

This can be treated as a tolerance design problem [11] if we take e_1 and e_2 to be the tolerances of the parameters x_1 and x_2, respectively. The optimum discrete solution is

$$\overset{v}{e}_1 = 0.4$$

$$\overset{v}{e}_2 = 0.5$$

$$\overset{vo}{x}_1 = 0.9046$$

$$\overset{vo}{x}_2 = 1.0060$$

$$\overset{v}{f} = 4.5001$$

The tree structure is shown in Figure 14. The feasible and acceptable regions of the design are shown in Figure 15.

CONCLUSIONS

A general user-oriented computer program for discrete value optimization is presented. We have assumed that all discrete variables in a problem are subject to the same quantization step. This can easily be relaxed by considering a quantization array Q, each element of which applies to a discrete variable, instead of a single quantization step q. Areas of application envisaged include digital filter design, tolerance considerations, and other computer-aided circuit design problems such as the design of lumped-element circuits where discrete values of components are often sought.

Parameter constraints are added or replaced at each stage, hence a current point is most likely to be a nonfeasible starting point for the following minimization process. The Bandler-Charalambous constraint transformation technique is therefore particularly advantageous in this case because the method does not require a feasible starting point. If other transformations, such as the Fiacco-McCormick technique, were used, much effort might be wasted in searching for a feasible starting point at each stage.

In spite of the fact that the program works quite satisfactorily, the efficiency of the algorithm is not exceedingly attractive because of the large number of function evaluations required for each minimization. For unimodal functions, a new scheme based on the general branch and bound technique and the validity of Kuhn-Tucker relations at an optimum point is under investigation. Figure 16 illustrates how the scheme works for a simple two-parameter problem.

The scheme starts by finding the continuous solution O. Then the four nearest lattice points, with parameter constraints as shown, are examined. It is obvious that the Kuhn-Tucker relations are satisfied for all but one point which is C. Let C be defined by (x_1^c , x_2^c). The objective function value increases along $x_1 = x_1^c$ but decreases along $x_2 = x_2^c$, hence the next lattice point E along $x_2 = x_2^c$ is checked. Kuhn-Tucker conditions are satisfied and the search is terminated. The optimum discrete solution is then selected by comparing the four feasible discrete solutions A, B, D and E. Extension of the scheme to n-parameter problems would be the next objective.

ACKNOWLEDGEMENT

Many useful suggestions from B.L. Bardakjian, C. Charalambous and P.C. Liu are acknowledged.

REFERENCES

[1] H.A. Taha, Operations Research --- An Introduction. New York: MacMillan, 1971.

[2] C. McMillan, Jr., Mathematical Programming. New York: Wiley, 1970.

[3] R.J. Dakin, "A tree-search algorithm for mixed integer programming problems", Computer J., vol. 8, pp. 250-255, 1966.

[4] L.G. Mitten, "Branch and bound methods: general formulations and properties", Operations Research, vol. 18, pp. 24-34, 1970.

[5] E.L. Lawler and D.E. Wood, "Branch and bound methods: a survey", Operations Research, vol. 14, pp. 699-719, 1966.

[6] R. Fletcher and M.J.D. Powell, "A rapidly convergent descent method for minimization", Computer J., vol. 6, pp. 163-168, 1963.

[7] R. Fletcher, "A new approach to variable metric

algorithms", Computer J., vol. 13, pp. 317-322, 1970.

[8] J.W. Bandler and C. Charalambous, "A new approach to nonlinear programming", Proc. 5th Hawaii Int. Conf. on System Sciences (Honolulu, Hawaii, Jan. 1972), pp. 127-129.

[9] J.N. Siddall, "OPTISEP-Designers' optimization subroutines", Faculty of Engineering, McMaster University, Hamilton, 1971 (revised).

[10] J.W. Bandler, J.H.K. Chen and V.K. Jha, "CONOPT-A package for solving nonlinear programming problems using a new (minimax) approach with efficient gradient methods", Department of Electrical Engineering, McMaster University, Hamilton, 1973.

[11] J.W. Bandler, "Optimization of design tolerances using nonlinear programming", Proc. 6th Princeton Conf. Information Sciences and Systems, (Princeton, N.J., March 1972), pp. 655-659.

No.	x_1	x_2	x_3	f
1	0	0	0	9
2	1	0	0	3
3	0	1	0	5
4	1	1	0	1
5	2	0	0	1
6	0	2	0	5
7	2	1	0	1
8	1	2	0	3
9	3	0	0	3
10	0	3	0	9
11	0	0	1	6
12	1	0	1	2
13	0	1	1	2

Table I.

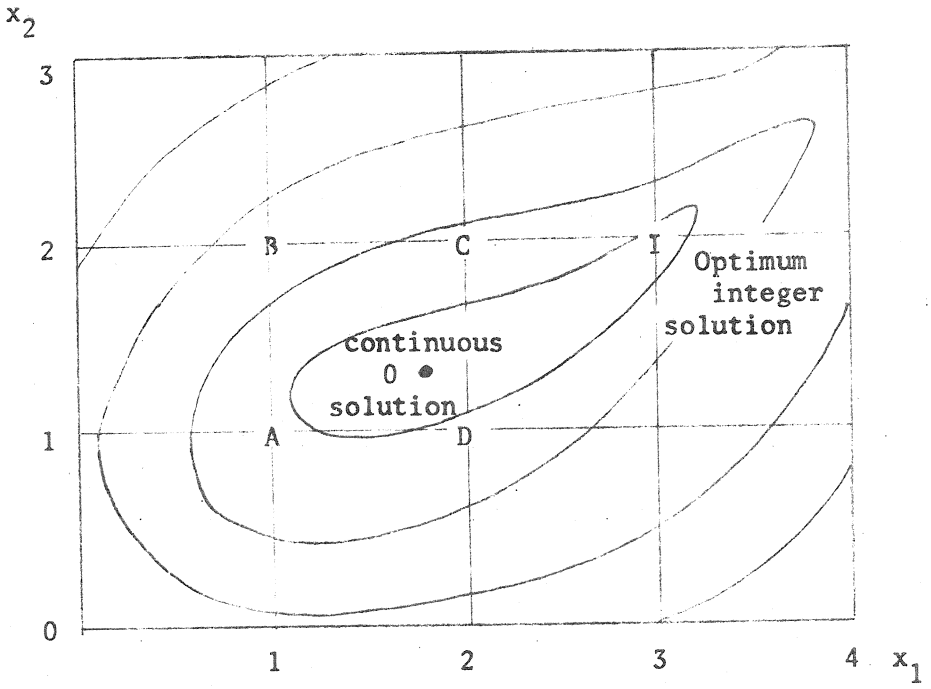

Figure 1. Optimum integer solution not obtainable by rounding off or truncating the optimum continuous solution in this case.

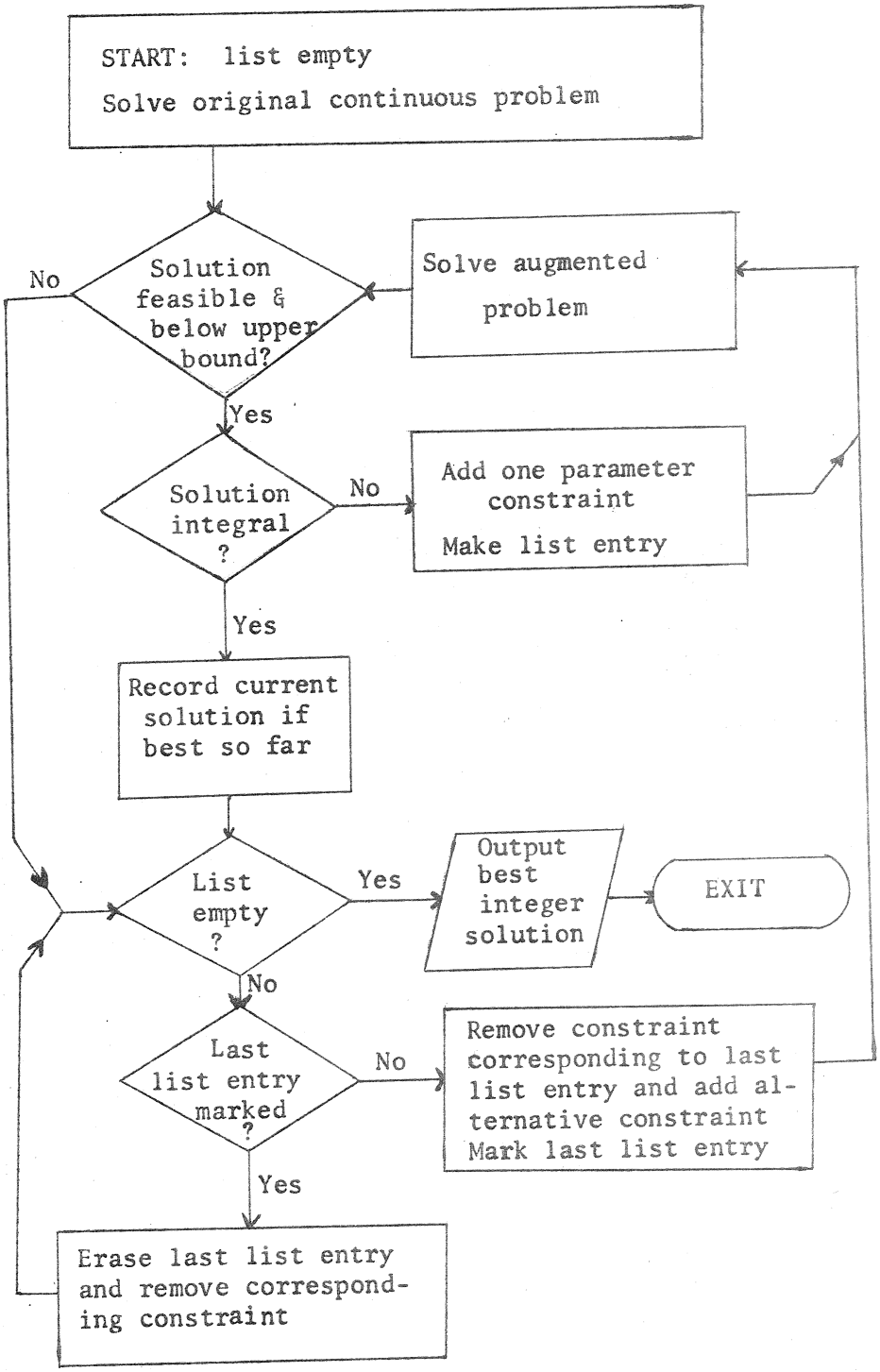

Figure 2. Flow diagram for Dakin's algorithm.

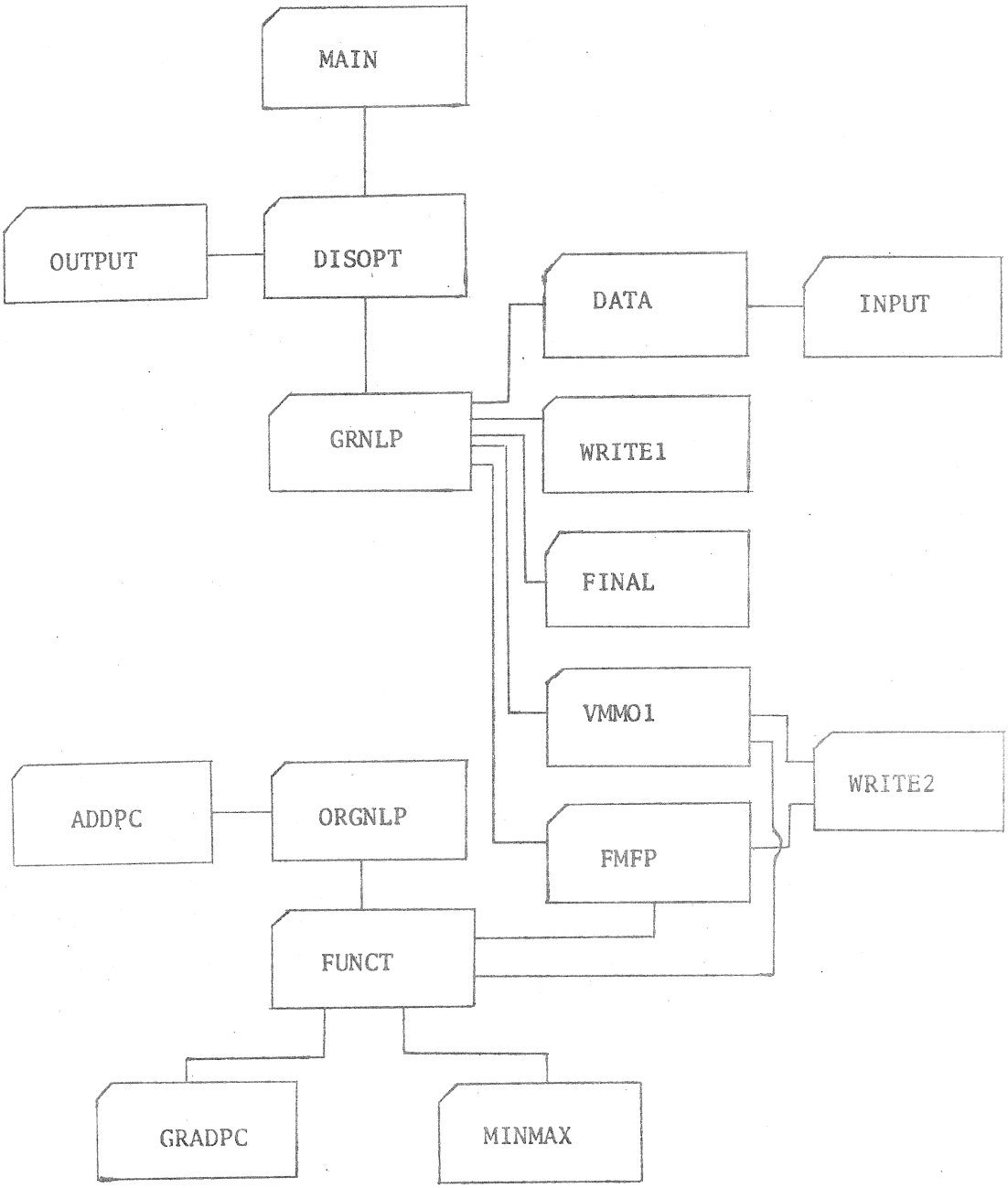

Figure 3. Overall structure of the package.

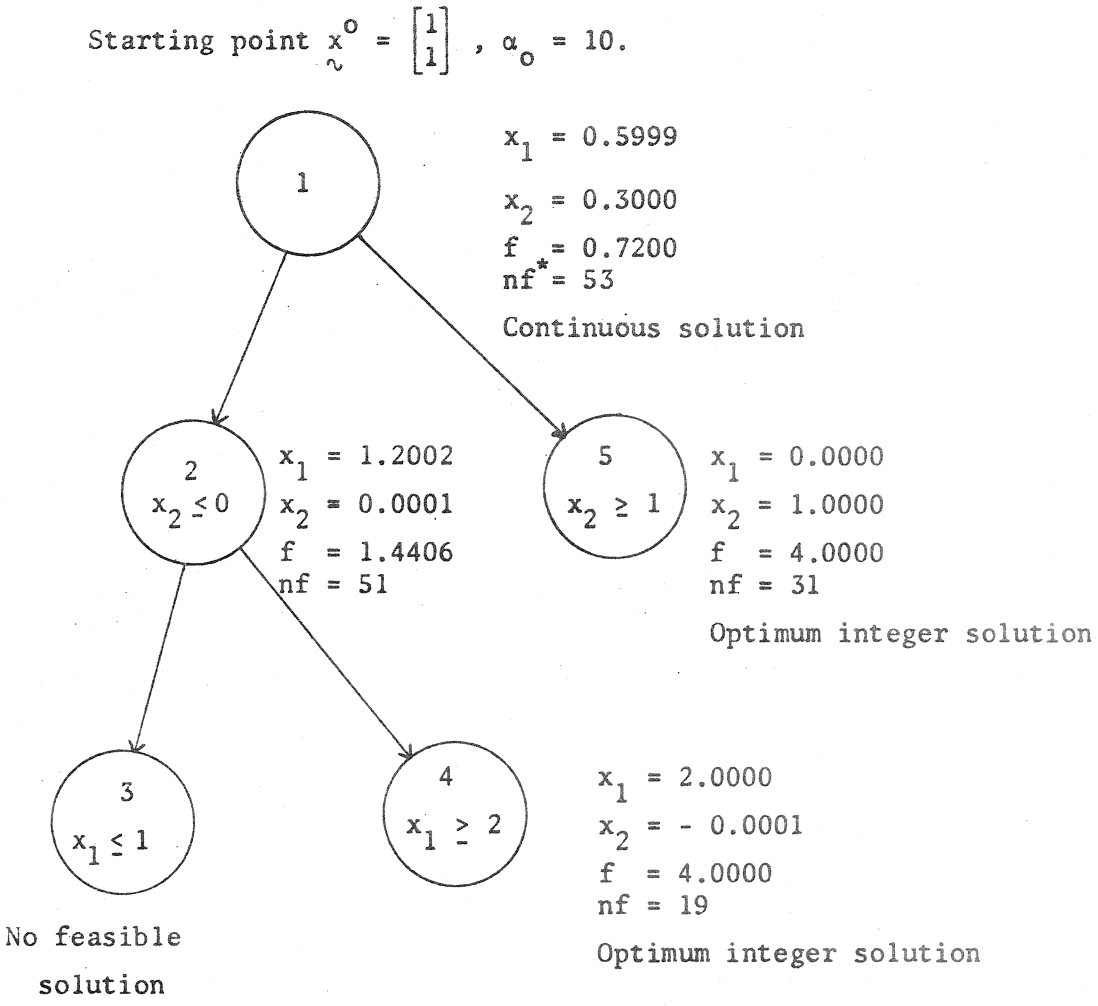

Figure 4. Tree for Example 1: q = 1. The Fletcher method was used (EPS = 10^{-4}).

---

*nf stands for the number of function evaluations taken in the minimization process

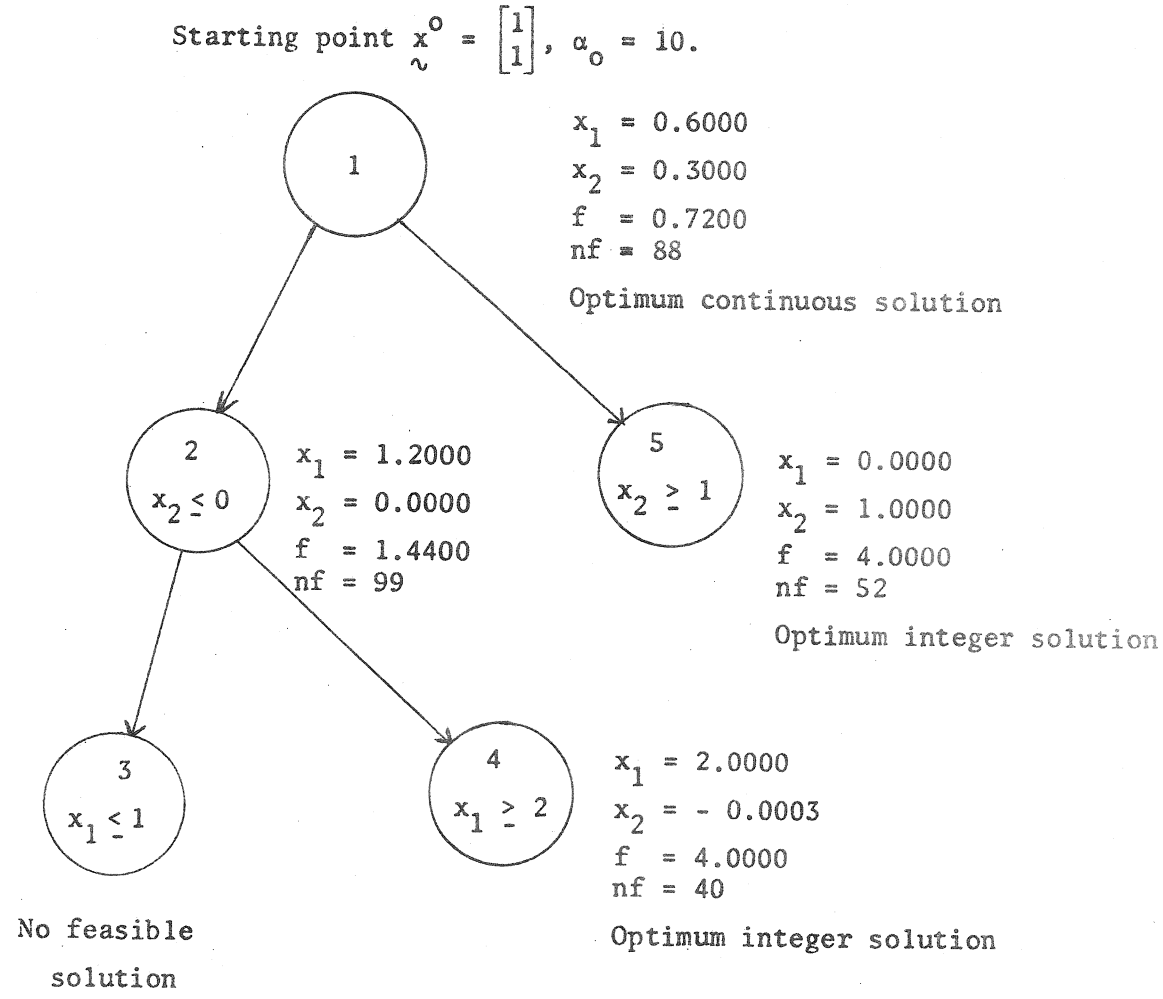

Figure 5. Tree for Example 1: q = 1. The Fletcher-Powell method was used (EPS1 = 10^{-5}). More function evaluations are required by the Fletcher-Powell method.

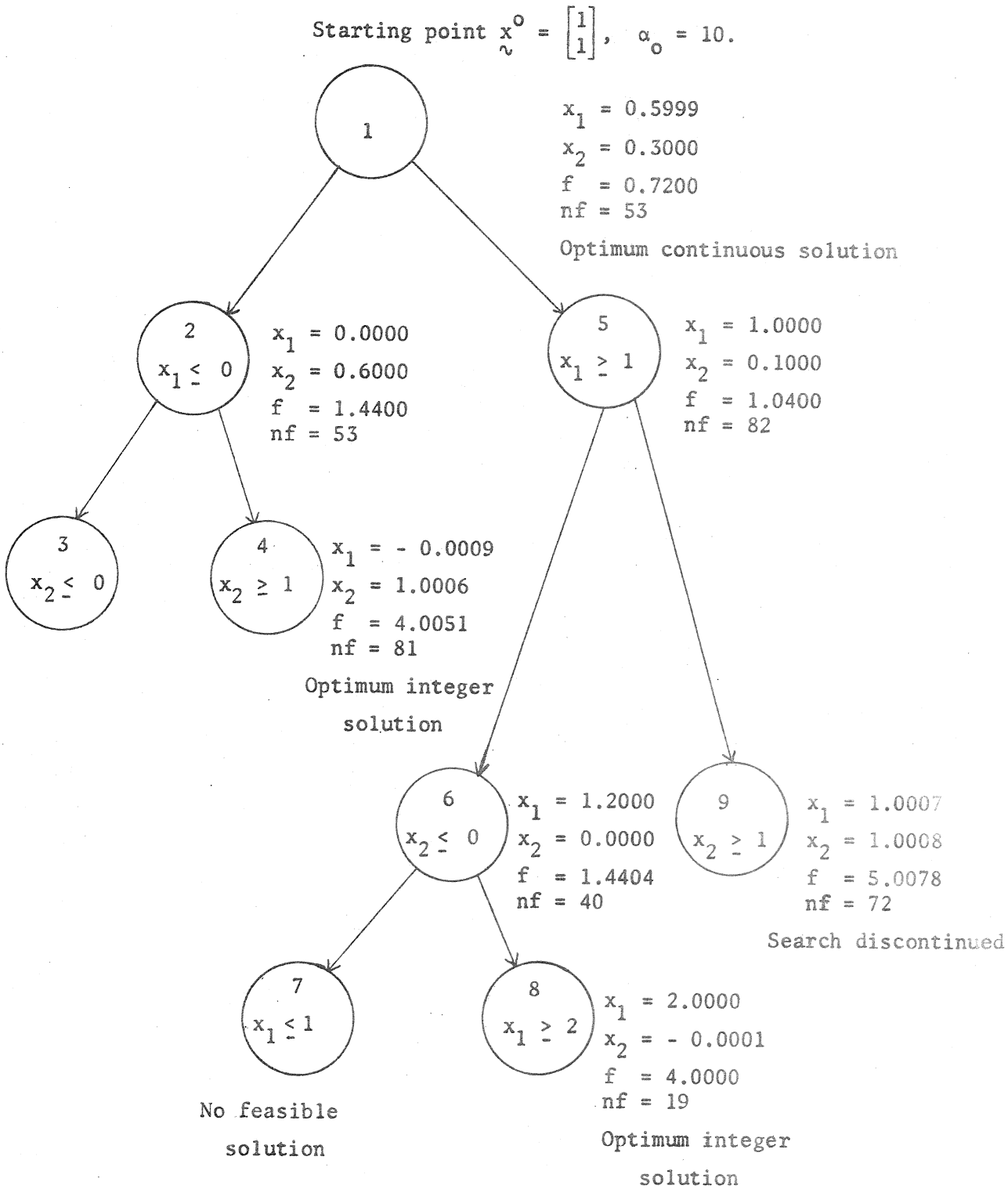

Figure 6. Tree for Example 1: q = 1. The Fletcher method was used (EPS = 10^{-4}).

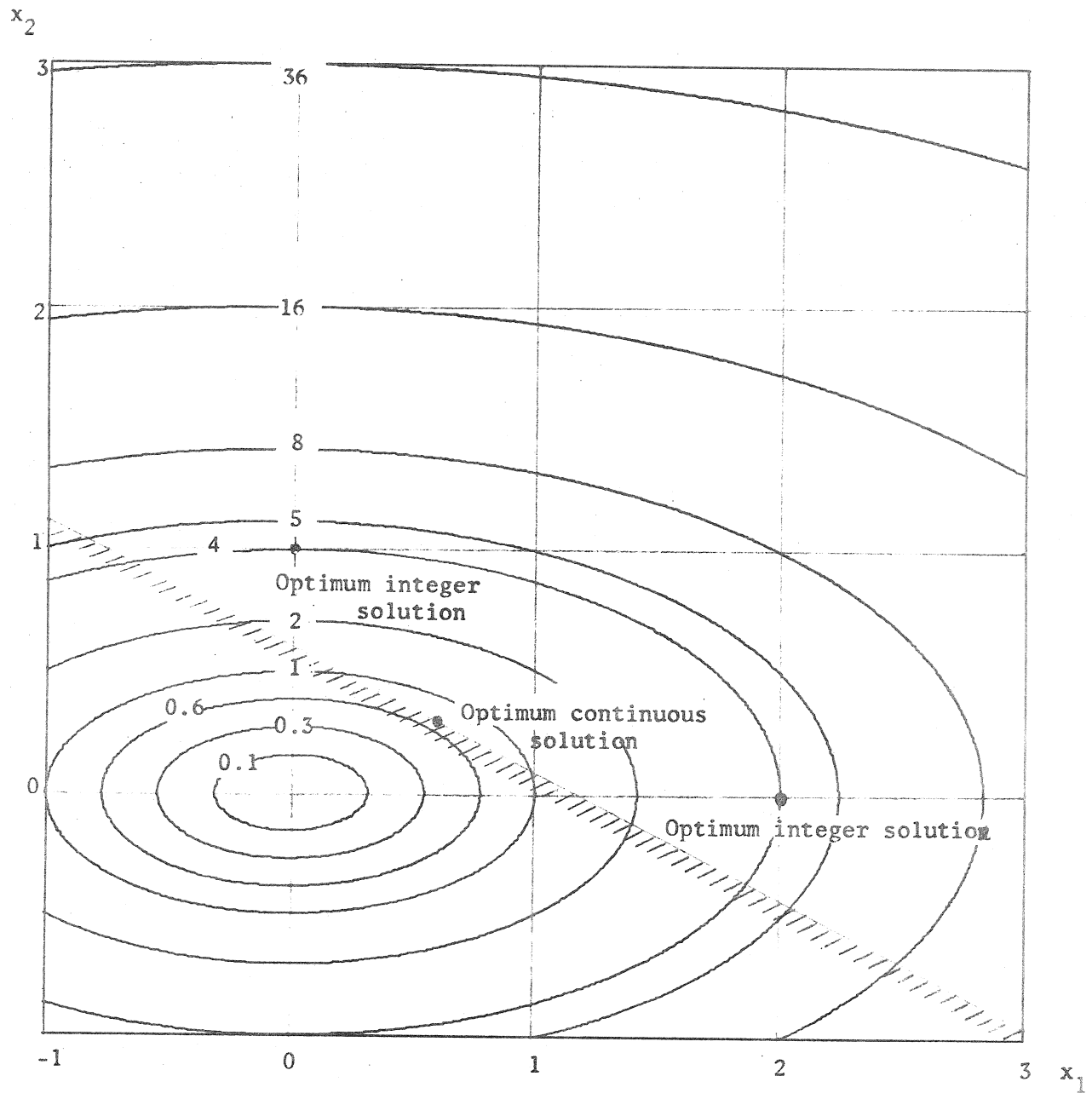

Figure 7. Contour plot for Example 1.

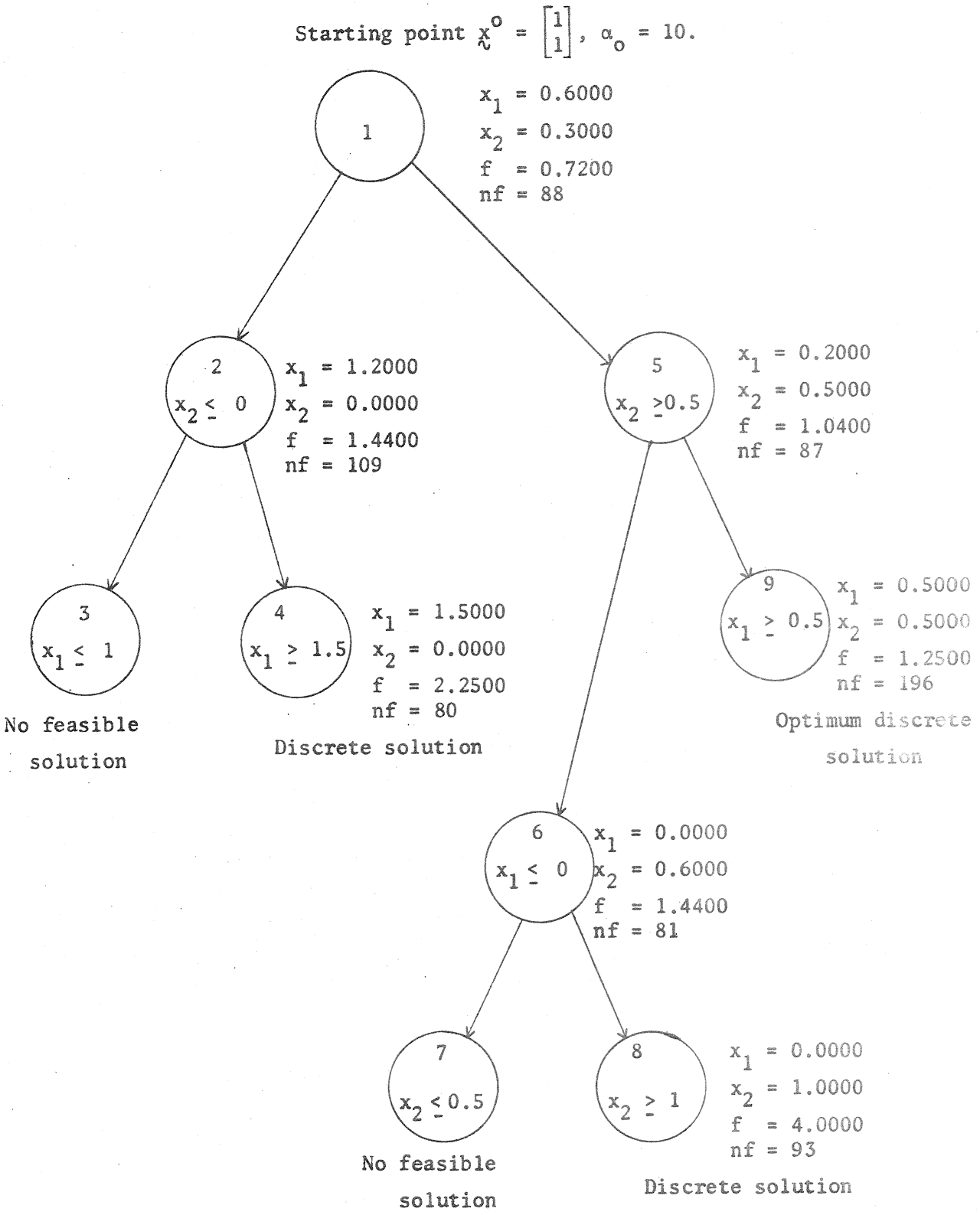

Figure 8. Tree for Example 1: q = 0.5. The Fletcher-Powell method was used (EPS1 = 10^{-6}).

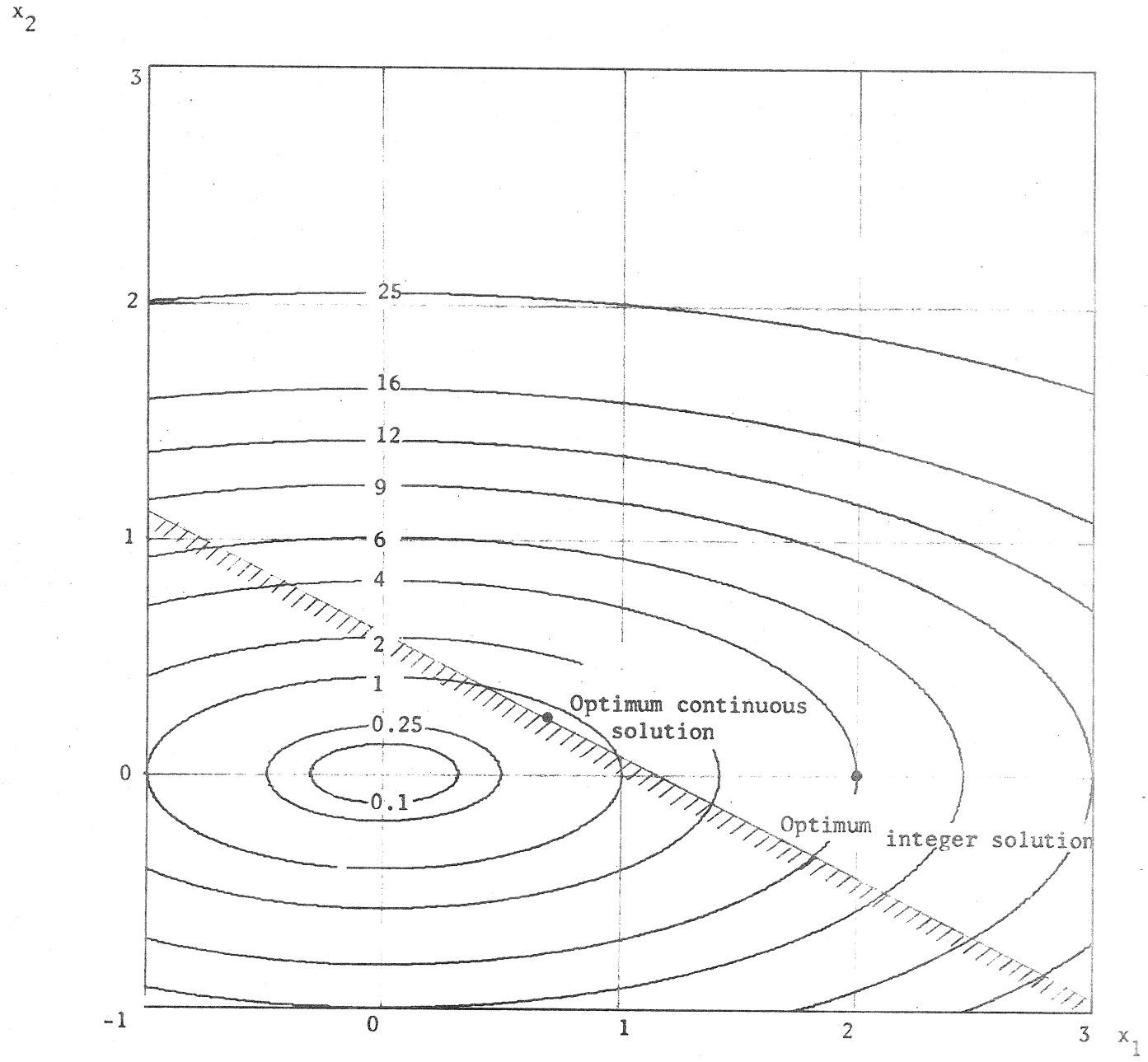

Figure 9. Contour plot for Example 2.

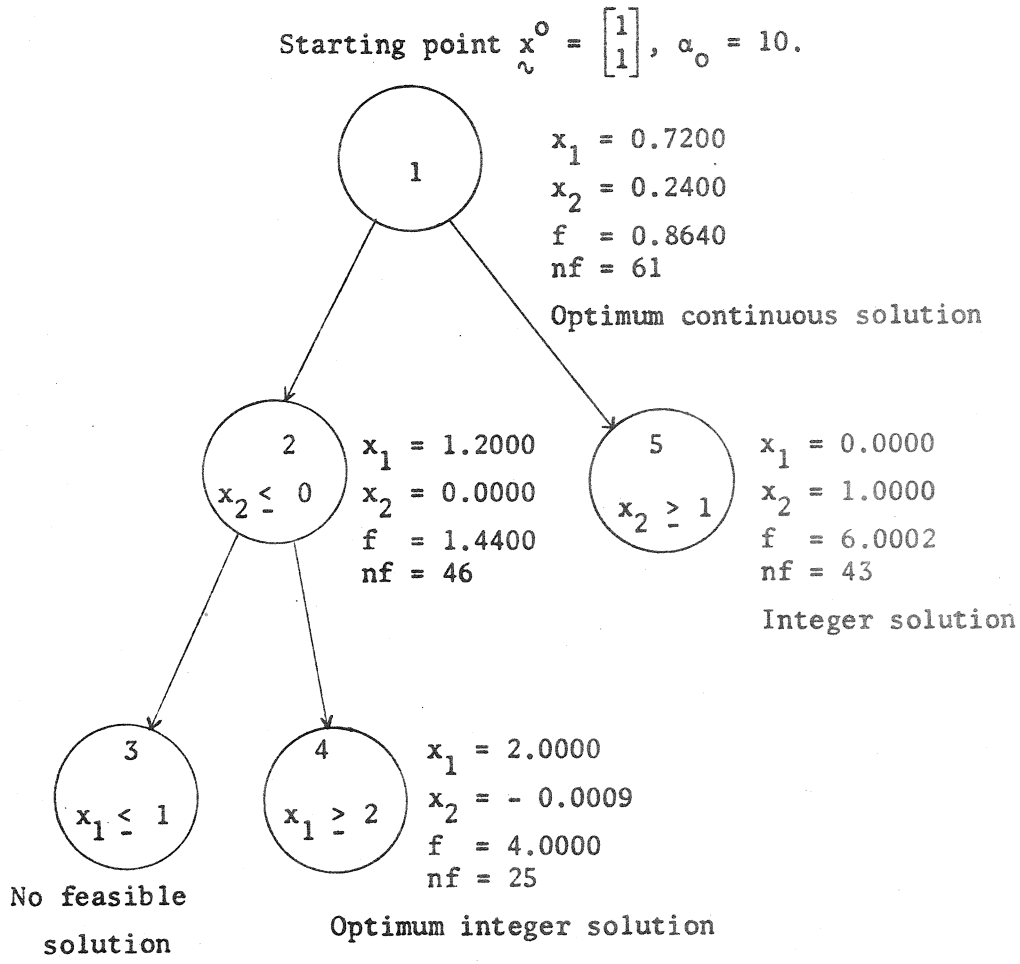

Figure 10. Tree for Example 2: q = 1. The Fletcher-Powell method was used (EPS = 10^{-4}).

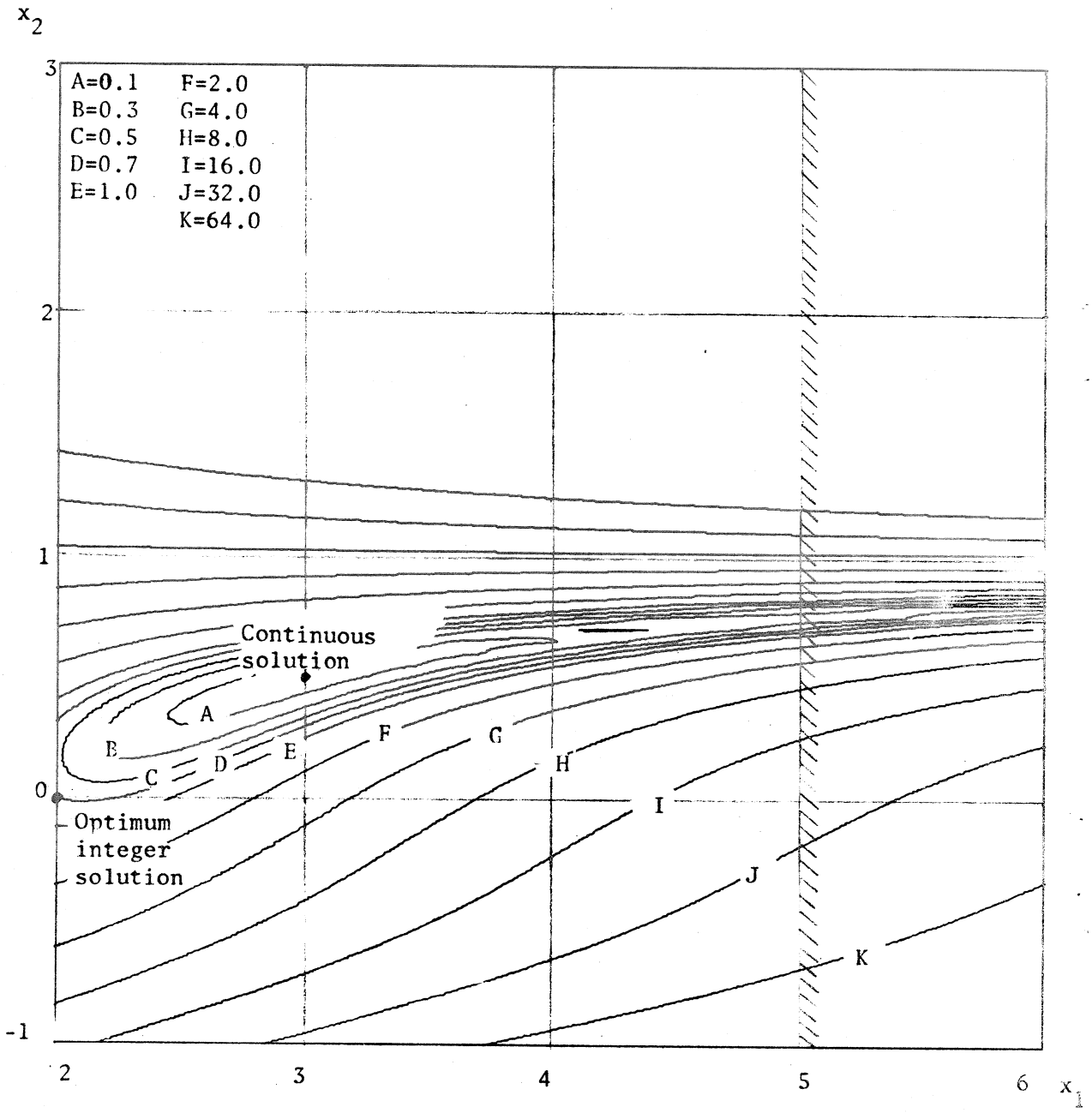

Figure 11. Contour plot for Example 3.

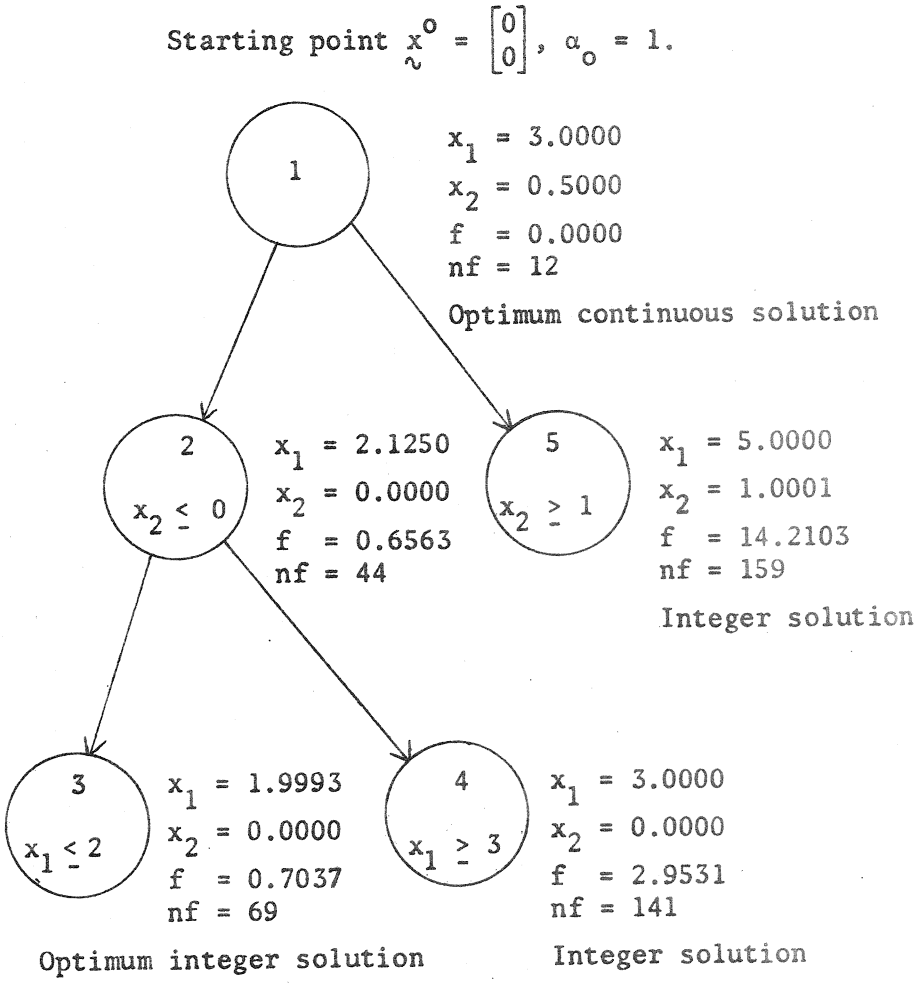

Figure 12 Tree for Example 3: $q = 1$. The Fletcher method was used (EPS1 = 10^{-4}).

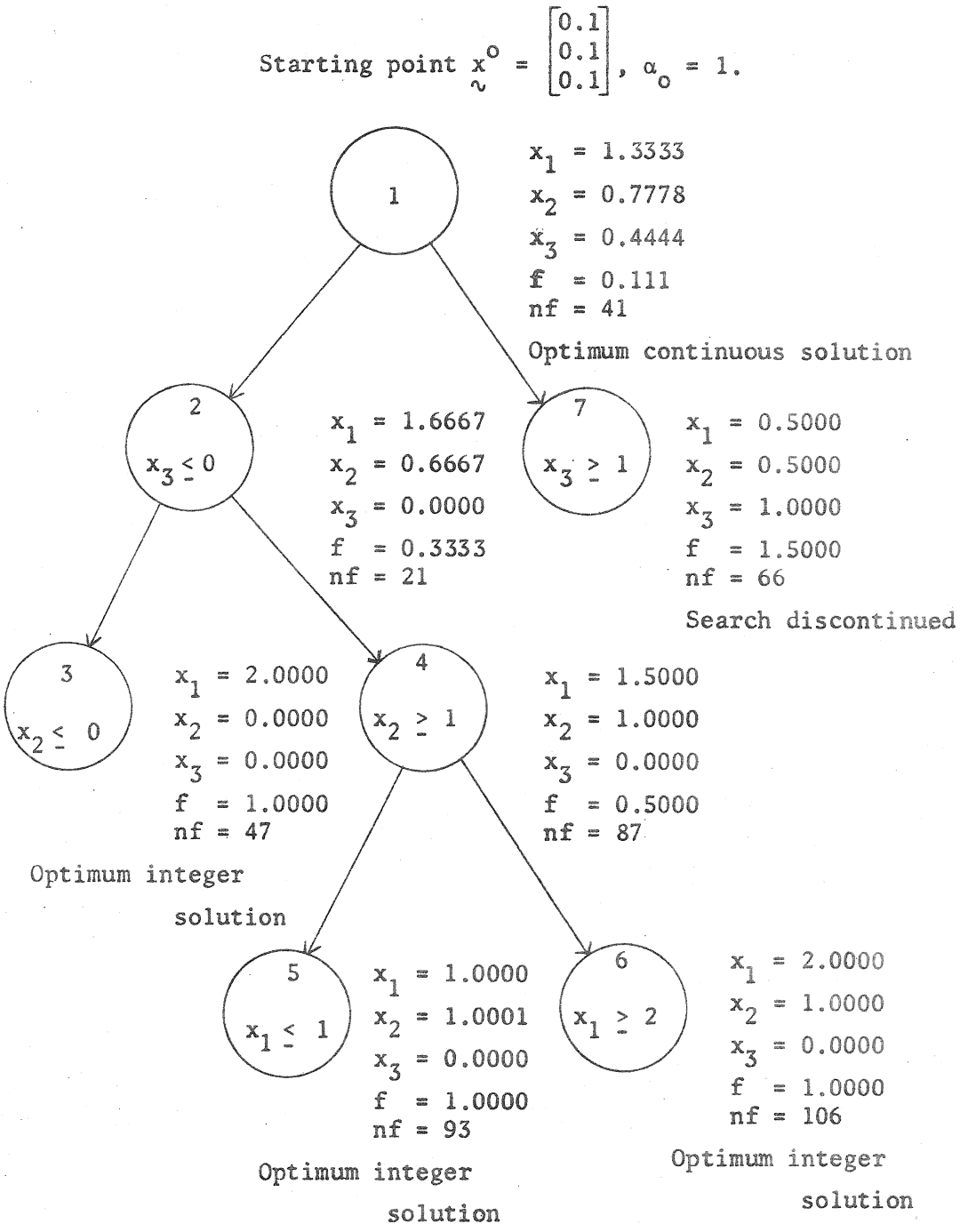

Figure 13. Tree for Example 4: q = 1. The Fletcher method was used (EPS1=10^{-4}). The maximum tolerable error in integer values is 0.001.

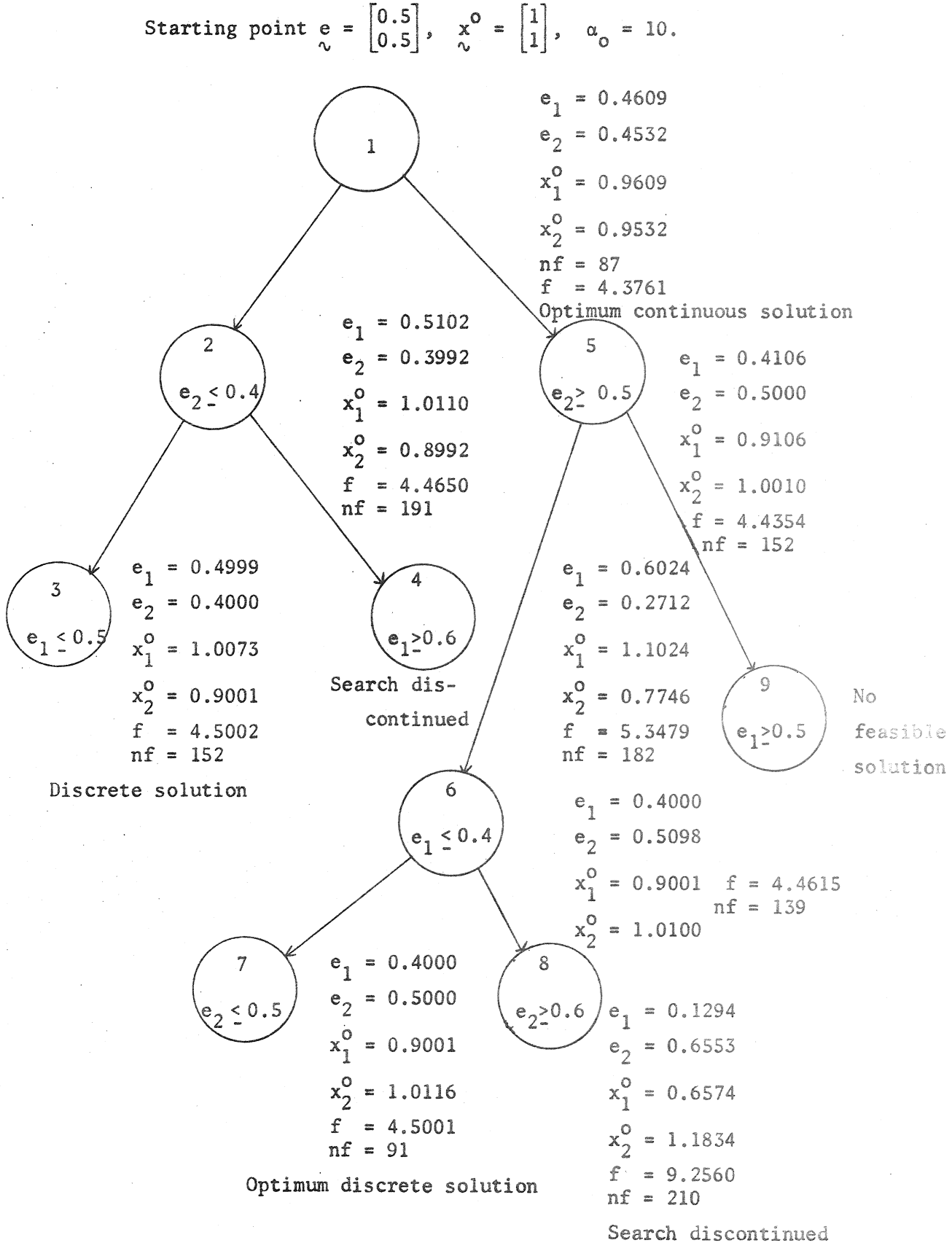

Figure 14. Tree for Example 5: q = 0.1. The Fletcher method was used (EPS1 = 10^{-4}).

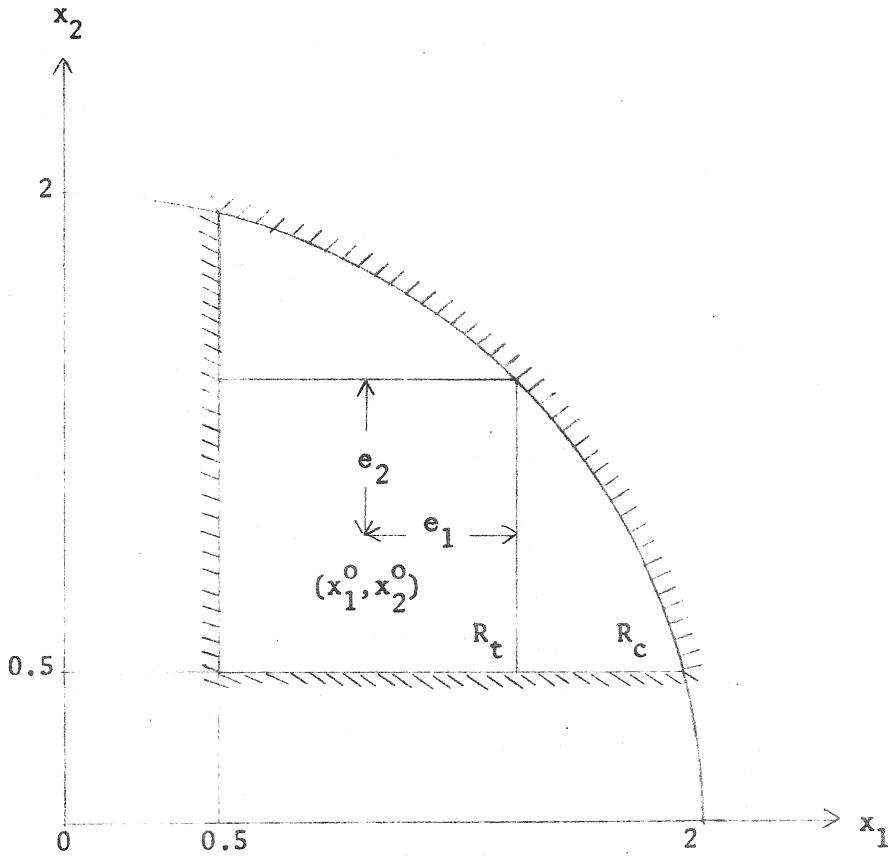

Figure 15. Feasible and acceptable design regions, R_c and R_t, for Example 5.

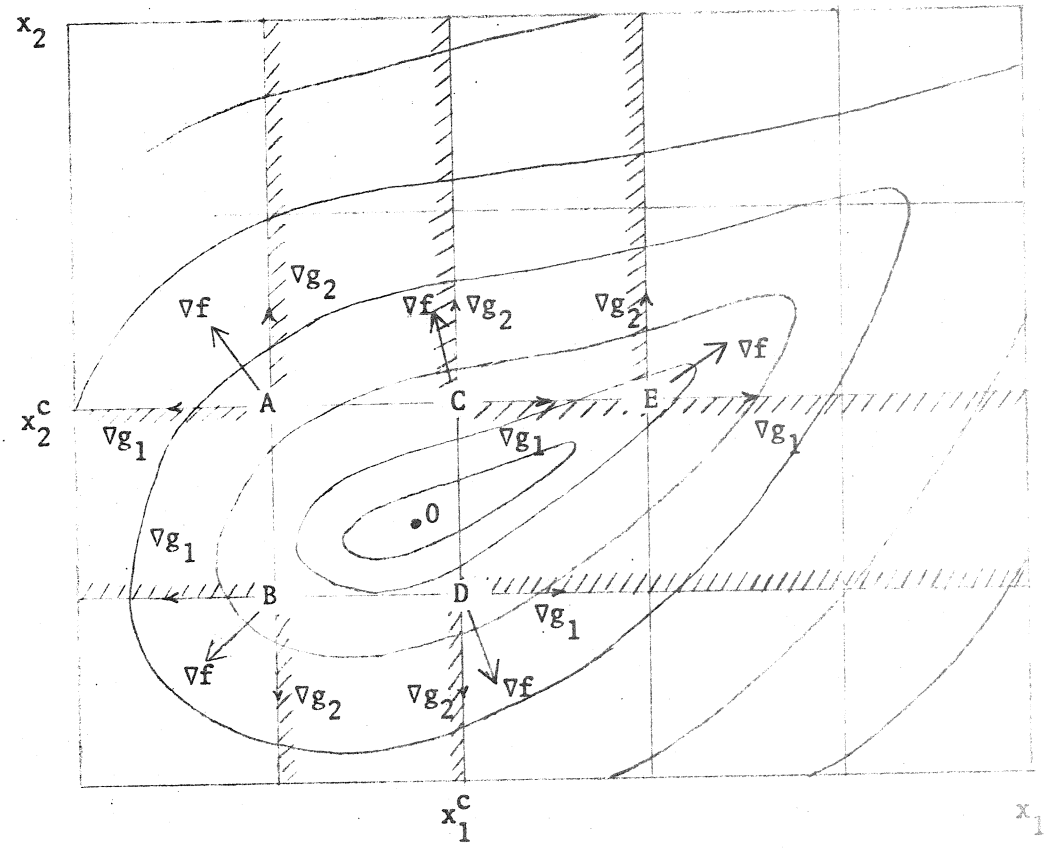

Figure 16. New scheme for discrete optimization.

APPENDIX

Control Cards

Use the following set of control cards.

```
JOBCARD
ATTACH, TAPE, DISOPT, ID=*********, MR=1*.
FTN, R=3, PL=30000B, ROUND = + - * /.
LOAD (TAPE)
LGO.
END OF RECORD
PROGRAM TST (INPUT, OUTPUT, TAPE5=INPUT, TAPE6=OUTPUT)
```

Main Program

Dimension the following arrays:

X(N), G(N), PE(N), XSTRT(N), DUM1(N), DUM2(N), EPS(N), H(J), IX(N), DIF(N), XB(N), PHI(NCONSP)

WHERE N = the number of independent variables

J = N(N + 7)/2

NCONSP = the total number of constraints anticipated

Supply the follwing COMMON statements:

```
COMMON /DDD/ NCONS
COMMON /KKK/ K
```

Supply the values of the following parameters:

---

* Appropriate identification parameter ID should be inserted in *********.

N = the number of independent variables

K = the number of discrete variables

NCONS = the number of original constraints

QSTEP = the quantization step

Call subroutine DISOPT and OUTPUT as follows:

```
CALL DISOPT(N, XSTRT, F, G, QSTEP, U, X, PHI, IX, DIF,
            XB, PE, DUM1, DUM2, EPS, H)
CALL OUTPUT(U, X, PHI, N)
```

Add STOP and END cards.

Subroutines ORGNLP and FUNCT

```
SUBROUTINE ORGNLP(X, PHI, IAA, IBB, U)
DIMENSION X(1), PHI(1), IAA(NCONSP), IBB(NCONSP)
```

$U = f(x_1, x_2, \ldots, x_n)$ is the actual objective function

$PHI(1) = g_1(x_1, x_2, \ldots, x_n)$

$PHI(2) = g_2(x_1, x_2, \ldots, x_n)$

.
.
.

$PHI(NCONS) = g_{NCONS}(x_1, x_2, \ldots, x_n)$ are the original inequality constraints

```
CALL ADDPC( X, PHI, IAA, IBB)
RETURN
END
```

Remark An equality constraint, $h(x_1, x_2, \ldots, x_n) = 0$ has to be treated as two inequality constraints, $h(x_1, x_2, \ldots, x_n) \geq 0$ and $h(x_1, x_2, \ldots, x_n) \leq 0$.

```
      SUBROUTINE FUNCT (N, X, F, G)
      DIMENSION X(1), G(1), IAA(NCONSP), IBB(NCONSP), GU(N),
     1          PHI(NCONSP), GPHI(N,NCONSP)
      COMMON /DDD/ NCONS
      COMMON /HHH/ NOD, KK
      COMMON /RRR/ NORG
      CALL ORGNLP (X, PHI, IAA, IBB, U)
      GU(1) = partial derivative of F w.r.t. x1
        .
        .
        .
      GU(N) = partial derivative of F w.r.t. xn
      GPHI(1, 1) = partial derivative of PHI(1) w.r.t. x1
          .
          .
          .
      GPHI(N, NCONS) = partial derivative of PHI(NCONS) w.r.t. xn
      IF (KK .EQ. 0) GO TO 1
      CALL GRADPC (N, GPHI, IAA, IBB)
    1 P = the power to be used in the least pth approximation
          (set equal to 1.E5 if no information is available)
      CALL MINMAX (U, GU, PHI, GPHI, N, NCONS, F, G, P, EPSPHI)
      RETURN
      END
```

Input Data

Parameters to be supplied as data are defined below.

METHOD	= 1, Fletcher method will be used = 2, Fletcher-Powell method will be used
MAX	Maximum number of permissible iterations
IPRINT	= 1, intermediate output is printed out for every iteration = 0, otherwise
IDATA	= 1, input data is printed out = 0, otherwise
MAXNOD	Maximum number of permissible nodes
ICON	= 1, first discrete variable is considered first = 0, last discrete variable is considered first
EST	Minimum estimated value of the objective function
EPS1	Small test quantity used by the Fletcher-Powell method
EPS(I), I=1, N	Test quantities used by the Fletcher method
XSTRT(I), I=1, N	Starting values for the variables.
AO	Initial value of the parameter alpha used in the formulation of the unconstrained

	objective function. It should be set equal to 1 if no information is available.
XMAL	Maximum permissible value of parameter alpha; no feasible solution assumed if alpha exceeds this value
ZERO	The margin by which constraints may be violated
ERR	Maximum tolerable error in discrete values

The Data Deck

Card No.	Format	Parameters
1	6I5	METHOD, MAX IPRINT, IDATA, MAXNOD, ICON
2	4E16.8	EST, EPS1, AO, XMAL
As many as required	5E16.8	(EPS(I), I=1, N)
As many as required	5E16.8	(XSTRT(I), I=1, N)
Last	2E16.8	ZERO, ERR

Note

The program is currently limited to 100 constraints.