WILLIAM CONLEY
University of Wisconsin-Green Bay

Computer Optimization Techniques

a petrocelli book
new york / princeton

Printed in the United States

1 2 3 4 5 6 7 8 9 10

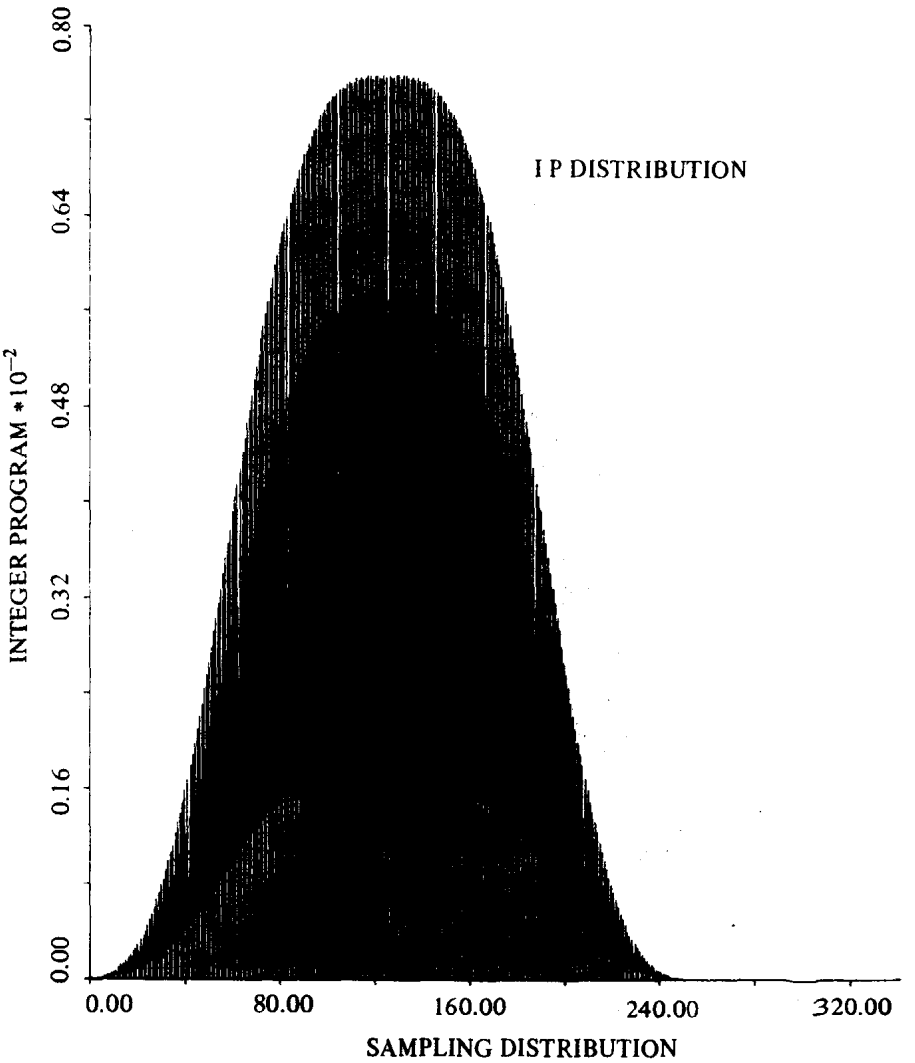
Designed by Joan Greenfield

Library of Congress Cataloging in Publication Data

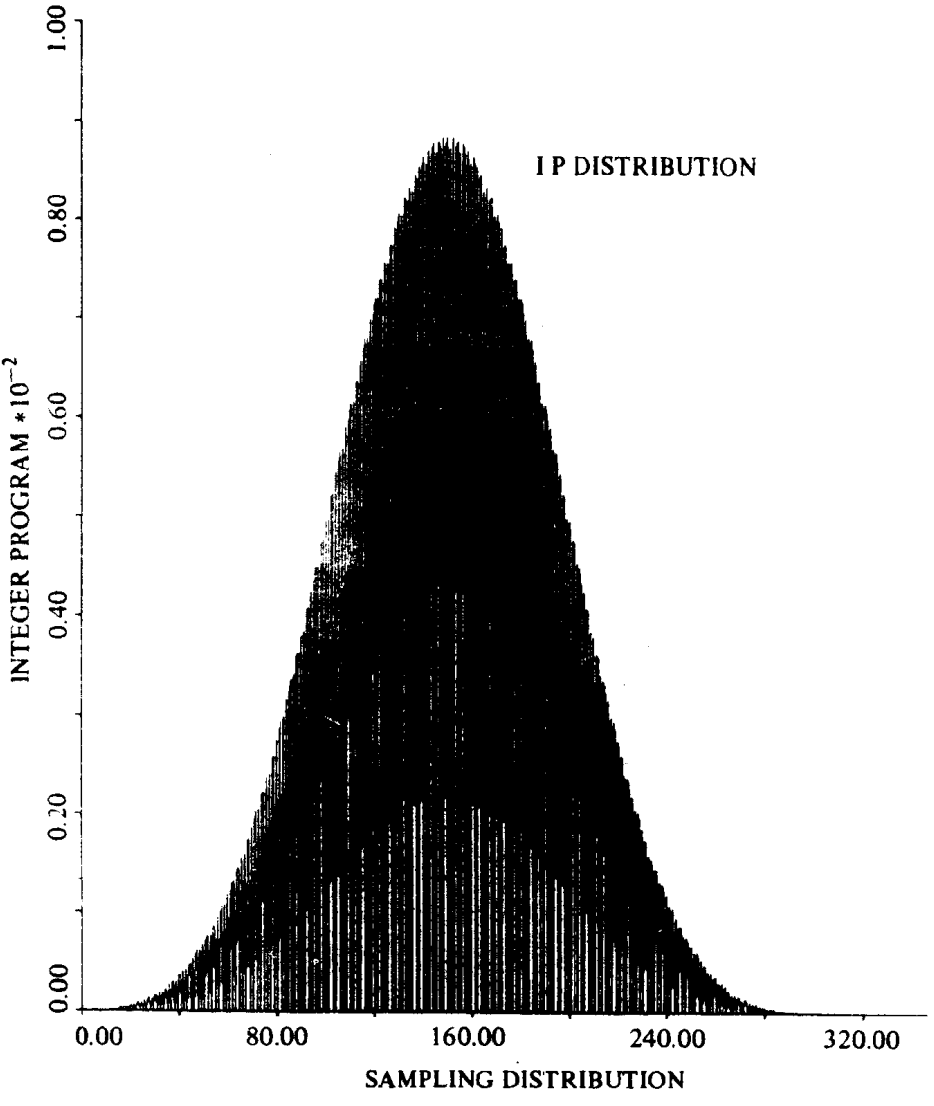
Conley, William, 1948–
Computer optimization techniques.

Includes index.
1. Mathematical optimization—Data processing.
2. Integer programming—Data processing. I. Title.
QA402.5.C64 519.7'7 79-24452
ISBN 0-89433-111-6

CONTENTS



of $P = x_1 + 3x_2 + 4x_3 + 5x_4 + 12x_5$ subject to $0 \leqslant x_i \leqslant 10 \quad i = 1, 5$



of $P = 4x_1 + 5x_2 + 6x_3 + 7x_4 + 8x_5$ subject to $0 \leqslant x_i \leqslant 10 \quad i = 1, 5$

PART ONE

CHAPTER 1

Optimization in the Computer Age

Mathematically, at least in our context, optimization means to find the maximum of a function or process that we want to maximize or to find the minimum of a function or process that we want to minimize. For example, we might wish to maximize a profit function or an output function of a process. Or, we might wish to minimize a cost function. Let's look at a few examples.

Suppose a company manufactures two products, A and B. Let x be the number of units of A produced and y the number of units of B produced. Suppose further that each unit of A returns a profit of two dollars and each unit of B returns a profit of three dollars. Therefore, the profit function would be written

$$P = 2x + 3y$$

where P is the profit in dollars.

Now, the question might naturally arise, how do we maximize this equation? Well, as stated the equation allows any values for x and y, therefore, it is only necessary to produce as much of A and B as possible to maximize P. P becomes infinitely large as either x or y or both go to infinity.

However, let's add a few restrictions to the variables x and y. Let's assume that the company's position is such that x must be between 0 and 10 inclusive, and y must be between 0 and 10 inclusive. In symbols this is $0 \leq x \leq 10$ and $0 \leq y \leq 10$. Let's further assume that x and y can only take integer values. This means that each possible x and y value must be a counting number or the negative of a counting number or zero. Equations to be optimized whose solution coordinates are restricted to integers (usually nonnegative integers in practical problems) are called integer programming problems. If we allow solutions that are not integer valued, like $x = .666$, $y = 7.5$, then we have a linear programming problem or a nonlinear noninteger programming problem. We, of course, can have a nonlinear integer programming problem. This is a problem in which either the function to be optimized and/or the constraints (conditions or restrictions) on the variables are nonlinear (they have squared and cubed terms, etc.). Also, in a nonlinear integer programming problem only integer coordinate solutions are allowed.

This book will deal mainly with integer programming problems (whole number coordinates for the solutions). But, fortunately, most applied problems require integer solutions. These are more difficult and sometimes almost impossible to solve theoretically. Later we hope to present a case for using integer solutions even in most cases where noninteger solutions are acceptable.

Getting back to our function to maximize, let us state the integer programming problem as maximize $P = 2x + 3y$ subject to $0 \leq x \leq 10$, $0 \leq y \leq 10$, and x and y must be integers. Therefore, let's look at the x and y pairs that are possibilities for the optimum. The following points are the only ones that satisfy the constraints:

(0,0) (1,0) (2,0) (3,0) (4,0) (5,0) (6,0) (7,0) (8,0) (9,0) (10,0)
(0,1) (1,1) (2,1) (3,1) (4,1) (5,1) (6,1) (7,1) (8,1) (9,1) (10,1)
(0,2) (1,2) (2,2) (3,2) (4,2) (5,2) (6,2) (7,2) (8,2) (9,2) (10,2)
(0,3) (1,3) (2,3) (3,3) (4,3) (5,3) (6,3) (7,3) (8,3) (9,3) (10,3)
(0,4) (1,4) (2,4) (3,4) (4,4) (5,4) (6,4) (7,4) (8,4) (9,4) (10,4)
(0,5) (1,5) (2,5) (3,5) (4,5) (5,5) (6,5) (7,5) (8,5) (9,5) (10,5)
(0,6) (1,6) (2,6) (3,6) (4,6) (5,6) (6,6) (7,6) (8,6) (9,6) (10,6)
(0,7) (1,7) (2,7) (3,7) (4,7) (5,7) (6,7) (7,7) (8,7) (9,7) (10,7)
(0,8) (1,8) (2,8) (3,8) (4,8) (5,8) (6,8) (7,8) (8,8) (9,8) (10,8)
(0,9) (1,9) (2,9) (3,9) (4,9) (5,9) (6,9) (7,9) (8,9) (9,9) (10,9)
(0,10) (1,10) (2,10) (3,10) (4,10) (5,10) (6,10) (7,10) (8,10) (9,10) (10,10)

Let's solve this problem by listing the 121 possible ordered pairs with their resultant P value in each case and then merely select the one that gives the largest value for P.

Possible points (amounts of A and B to be made) are sometimes called feasible solutions.

Points	$P = 2x + 3y$ value
(0,0)	0
(1,0)	2
(2,0)	4
(3,0)	6
(4,0)	8
(5,0)	10
(6,0)	12
(7,0)	14
(8,0)	16
(9,0)	18
(10,0)	20
(0,1)	3
(1,1)	5
(2,1)	7
(3,1)	9
(4,1)	11
(5,1)	13
(6,1)	15
(7,1)	17
(8,1)	19
(9,1)	21
(10,1)	23
(0,2)	6
(1,2)	8
(2,2)	10
(3,2)	12
(4,2)	14
(5,2)	16
(6,2)	18
(7,2)	20
(8,2)	22
(9,2)	24
(10,2)	26
(0,3)	9
(1,3)	11
(2,3)	13
(3,3)	15
(4,3)	17
(5,3)	19
(6,3)	21
(7,3)	23

Points (continued)	$P = 2x + 3y$ value (continued)
(8,3)	25
(9,3)	27
(10,3)	29
(0,4)	12
(1,4)	14
(2,4)	16
(3,4)	18
(4,4)	20
(5,4)	22
(6,4)	24
(7,4)	26
(8,4)	28
(9,4)	30
(10,4)	32
(0,5)	15
(1,5)	17
(2,5)	19
(3,5)	21
(4,5)	23
(5,5)	25
(6,5)	27
(7,5)	29
(8,5)	31
(9,5)	33
(10,5)	35
(0,6)	18
(1,6)	20
(2,6)	22
(3,6)	24
(4,6)	26
(5,6)	28
(6,6)	30
(7,6)	32
(8,6)	34
(9,6)	36
(10,6)	38
(0,7)	21
(1,7)	23
(2,7)	25
(3,7)	27

Points (continued)	$P = 2x + 3y$ value (continued)	Points (continued	$P = 2x + 3y$ value (continued)
(4,7)	29	(2,9)	31
(5,7)	31	(3,9)	33
(6,7)	33	(4,9)	35
(7,7)	35	(5,9)	37
(8,7)	37	(6,9)	39
(9,7)	39	(7,9)	41
(10,7)	41	(8,9)	43
(0,8)	24	(9,9)	45
(1,8)	26	(10,9)	47
(2,8)	28	(0,10)	30
(3,8)	30	(1,10)	32
(4,8)	32	(2,10)	34
(5,8)	34	(3,10)	36
(6,8)	36	(4,10)	38
(7,8)	38	(5,10)	40
(8,8)	40	(6,10)	42
(9,8)	42	(7,10)	44
(10,8)	44	(8,10)	46
(0,9)	27	(9,10)	48
(1,9)	29	(10,10)	50

We can see that, as expected, the optimum solution (the one that maximizes the profit is $x = 10$ units of A and $y = 10$ units of B.

This may seem like a lot of work to obtain this rather obvious result. However, it should be noted that conceptually it is an easy approach, namely, just examine all possible points. Also, it will always lead to the correct answer. This will be especially useful when the function to be maximized or minimized and/or the constraints are sufficiently complicated so that the solution is difficult to obtain either by inspection or through mathematical theory. This is frequently the case in applications.

Of course, the approach we take, namely, listing all possible solutions, is extremely tedious for people even though it is straightforward. However, a computer just loves repetitive, tedious work and will produce the answer in seconds. And as the speed and capacity of computers increase this technique will become more and more practical.

Let's look at another example. Try to minimize the cost equation $C = 2x^2 - y^2 + xy$ where x can take the values between 0 and 5 and y can take the values between 0 and 5, and x and y must be integers. The possible points meeting the constraints are as follows:

(0,0) (1,0) (2,0) (3,0) (4,0) (5,0)
(0,1) (1,1) (2,1) (3,1) (4,1) (5,1)
(0,2) (1,2) (2,2) (3,2) (4,2) (5,2)
(0,3) (1,3) (2,3) (3,3) (4,3) (5,3)
(0,4) (1,4) (2,4) (3,4) (4,4) (5,4)
(0,5) (1,5) (2,5) (3,5) (4,5) (5,5)

Let's list the possible points (combinations of x and y) along with the corresponding $C = 2x^2 - y^2 + xy$ value and take the points which produce the minimum. There are 36 possibilities:

Points	$C = 2x^2 - y^2 + xy$ value	Points (continued)	$C = 2x^2 - y^2 + xy$ value (continued)
(0,0)	0	(1,3)	−4
(1,0)	2	(2,3)	5
(2,0)	8	(3,3)	18
(3,0)	18	(4,3)	35
(4,0)	32	(5,3)	56
(5,0)	50	(0,4)	−16
(0,1)	−1	(1,4)	−10
(1,1)	2	(2,4)	0
(2,1)	9	(3,4)	14
(3,1)	20	(4,4)	32
(4,1)	35	(5,4)	54
(5,1)	54	(0,5)	−25
(0,2)	−4	(1,5)	−18
(1,2)	0	(2,5)	−7
(2,2)	8	(3,5)	8
(3,2)	20	(4,5)	27
(4,2)	36	(5,5)	50
(5,2)	56		
(0,3)	−9		

We can see that the optimum solution (the one that minimizes the cost) is $x = 0$ and $y = 5$. This yields a C value of -25.

Now, let's try to maximize $P = 3x^2 - 2y$ where x and y must be nonnegative integers and, further, they must satisfy $y \leq -.5x + 5$ and $y \leq -2x + 10$. A graph of the related equalities is given in Figure 1.1. The shaded region shows the area that satisfies the inequalities. Generally speaking, with an inequality of the form $y \leq mx + b$ the solution is the half plane below the line $y = mx + b$. This is the case here. Let's now list the integer combinations that satisfy the constraints along with their corresponding function values and take the coordinates that give us a maximum P under the constraints:

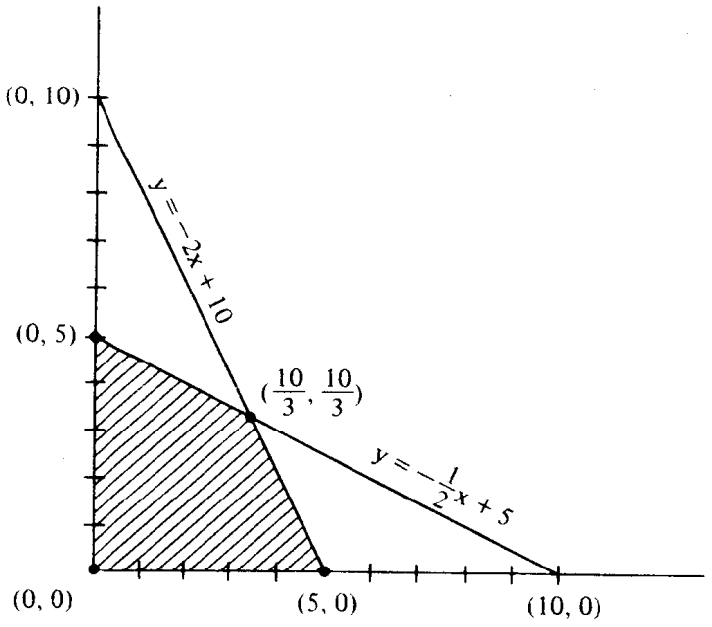


Figure 1.1

Points	$P = 3x^2 - 2y$ value	Points	$P = 3x^2 - 2y$ value
(0,0)	0	(1,2)	−1
(1,0)	3	(2,2)	8
(2,0)	12	(3,2)	23
(3,0)	27	(4,2)	44
(4,0)	48	(0,3)	−6
(5,0)	75	(1,3)	−3
(0,1)	−2	(2,3)	6
(1,1)	1	(3,3)	21
(2,1)	10	(0,4)	−8
(3,1)	25	(1,4)	−5
(4,1)	46	(2,4)	4
(0,2)	−4	(0,5)	−10

We can see that the maximum of P occurs at (5,0). We get a P value of 75 for that point. Therefore, this is the solution to the integer programming problem:

Maximize $P = 3x^2 - 2y$ subject to

$x \leq 0$, $y \leq 0$, $y \leq -2x + 10$ and $y \leq -.5x + 5$

Now, let's consider the integer programming system $P = 12x_1^2 + 17x_2^2 + 37x_3^2 + 18x_4^2 - 29x_5^2 + x_1x_3$ subject to

$$x_1 \geq 0, x_2 \geq 0, x_3 \geq 0, x_4 \geq 0, x_5 \geq 0$$
$$x_1 + x_2 + x_3 + x_4 + x_5 \leq 400$$
$$x_1 + x_2 + x_3 \leq 200$$
$$x_1 + x_4 + x_5 \leq 300$$
$$18x_1^2 + 17x_5 \leq 1000$$

Conceptually, we can take the same approach as before. This problem is extremely difficult, if not impossible, to solve theoretically. However, we could just list all the sets of five nonnegative integer points that satisfy the above constraints and evaluate P for each of these and take the set of five points that produces the largest value for the solution. So, conceptually, this is just as easy as the other examples although it might take dozens of hours to check and list all the right points. However, we can write a FORTRAN IV program in about five minutes that will easily search all of these points. And with the speed of computers increasing and the cost of computing time dropping dramatically, this is certainly one of the best approaches to take with these types of problems. It didn't used to be a practical approach but it is now.

We have available quite inexpensive minicomputers that sit around for hours each day not being used. They could be put to use solving these types of problems for virtually no cost at all, and the beauty of the approach of checking the points is that you always get an answer.

For those readers who may be thinking about what can be done if there are 20 or 200 variables with just too many feasible points (or feasible solutions) to check, Part Two will show how to get an answer to those problems, too.

Let's also keep in mind that we live in a world that increasingly puts constraints (inequalities, etc.) on our production or other operations. Therefore, it is more necessary to operate efficiently, and hence integer programming problems become very relevant and important. Also, this technique will increase even the efficiency of solving integer programming problems by making the very fast, efficient, and inexpensive computer do the inefficient part of the work. This will ultimately cut costs.

Review of FORTRAN IV

FORTRAN IV is probably the most popular and best version of FORTRAN, and it is the computer language that will be used in this book to solve integer programming problems. Therefore, it is recommended that the reader either be familiar with FORTRAN or, if not, become acquainted with it through one of the many fine texts available on the subject.

However, just for the purposes of review, and to try to illuminate our goals, let's discuss a few of the details of integer programming problems. We will also mention here that any language with loops, IF statements, and a random number generator would work in place of FORTRAN.

First of all, FORTRAN is short for *formula translation*. Its chief usefulness is its ability to evaluate any function or formula very quickly. It can also evaluate any formula for a large number of possible values in a very short time.

Therefore, let's assume that we have a profit function $P(x,y) = 7x^2 + 3x + 2xy - y^2 + 8y$ that depends on which combinations of x and y that we choose to use. Let's further assume that x can take the values 0, 1, 2 and 3, and y can take the values 0, 1, and 2. Therefore, there are $4 \times 3 = 12$ possible combinations that we can try in our effort to find the combination which yields the maximum profit.

The combinations are:

(x,y)
(0,0)
(0,1)
(0,2)
(1,0)
(1,1)
(1,2)
(2,0)
(2,1)
(2,2)
(3,0)
(3,1)
(3,2)

Now, it would take a bit of work to evaluate the profit function $P(x,y)$ for the twelve combinations of x and y. However, if we do this we can then compare the twelve resulting $P(x,y)$ values and note which one is the largest. We then can maximize our profit function by choosing the ordered pair (x and y combination) that yields this maximum profit value.

A computer programmed in FORTRAN could do the same thing we are attempting to do by hand, and it can do it about one million times faster than we can. So if the computer had to search 12 points or 12 million points for the maximum, it could do it quite easily. Also, with the revolution in minicomputers bringing 24 hours a day of computer power to the desk top of any manager, this idea of searching all possible points (or combinations) or a great many points is suddenly practical for the first time. The cost of this approach is essentially only the electricity required to run the minicomputers for the length of time necessary to find a good optimum value.

As an illustration, a typical integer programming problem that we worked out in the text checked 160,000 combinations of four different variables and calculated and stored the optimum solution in 18 seconds. The cost involved is so small it is hardly worth discussing, but the technique is.

Getting back to the profit function with the twelve possible combinations of x and

y, let's write a FORTRAN IV program to solve this question, while reviewing the rules of FORTRAN IV and keeping in mind that this approach can be adopted to any integer programming problem.

FORTRAN IV is a language that consists of a series of statements arranged vertically and executed sequentially from the top to the bottom, unless this order is interrupted by a command or control statement which orders the computer to proceed to a statement other than the next one. If this happens, the computer jumps to the statement as ordered and then proceeds sequentially until the direction of flow is changed by another command or it reaches the STOP and END statements at the conclusion of the program.

Any letter can be used for a variable. In fact, any sequence of up to six letters, numbers, and the dollar sign character can be used as variables. Equals signs are also used in FORTRAN IV. They order the computer to evaluate the function or relation on the right-hand side of the equals sign and assign that current value to the variable on the left. Also +, −, /, *, and ** are used for addition, subtraction, division, multiplication, and raising a variable to a power (exponentiation), respectively. Parentheses are used where necessary as in algebra. For example, if we wanted to evaluate $P(x,y) = 7x^2 + 3x + 2xy - y^2 + 8y$ for $x = 1$ and $y = 2$ and print x,y and $P(x,y)$, we could write it as follows in FORTRAN IV:

```
   X=1
   Y=2
   PXY=7*X**2+3*X+2*X*Y-Y**2+8*Y
   WRITE (6,3)X,Y,PXY
 3 FORMAT ('0',3E15.7)
```

Just controls the form of the output. Not essential to the logic (see a FORTRAN text).

This would cause the computer to output 1, 2, and 26 which is the value of P for $x = 1$ and $y = 2$.

Now we would like to write a FORTRAN IV program to systematically search all twelve combinations of x and y and compare the $P(x,y)$ values and store the maximum value along with its x and y coordinates until the end of the program, and then print the optimum xy combination and the corresponding largest $P(x,y)$ value.

We will now write a complete FORTRAN IV program to do this and then go over it, introducing the few new statements used in it. The idea is this: we initialize a variable (call it PMAX) to be some very small value such that $P(x,y)$ will be greater than that value for all possible combinations of x and y. Then, we go through all possible combinations of x and y storing the combinations and P (the $P(x,y)$ value) when they produce a $P(x,y)$ value larger than all the previous values. When the program is through, it prints the x and y combination that produced the maximum plus the maximum $P(x,y)$ value which has been stored in the variable PMAX.

The program for this example is as follows:

```
      INTEGER X,Y,PMAX,P
      PMAX=-999999
      DO 1 I=1,4
      X=I-1
      DO 1 J=1,3
      Y=J-1
      P=7*X**2+3*X+2*X*Y-Y**2+8*Y
      IF(P.GT.PMAX) GO TO 2
      GO TO 1
2     IX=X
      IY=Y
      PMAX=P
1     CONTINUE
      WRITE(6,3)IX,IY,PMAX
3     FORMAT('0',3I10)
      STOP
      END
```

Statement 1 declares X, Y, PMAX, and P to be integer variables. This means that only whole number values will be assigned to them. Any fractional or decimal part of the number will be dropped. However, the function deals only with whole numbers and its $P(x,y)$ values are only whole numbers for the 12 xy combinations, therefore, no accuracy is lost.

Statement 2 initializes PMAX to be an extremely small number so that the first calculated $P(x,y)$ value (called P in the program) will be larger than the initial PMAX value.

Statements 3 through 13 comprise a DO-loop. Essentially, this tells the computer to proceed sequentially from statement 3 to 13 (of course, jumping around as ordered, also) four times. In other words, with I = 1 the computer goes from 3 to 4 . . . 13. Then with I = 2 the computer goes from 3 to 4 . . . 13. Then with I = 3 the computer goes from 3 to 4 . . . 13. And finally with I = 4, the computer goes from 3 to 4 . . . 13. While this is taking place, there is another DO-loop inside the above outer one. This runs from statement 5 to statement 13. The effect of this loop is that it forces the computer to go from statement 5 to 13 (for J = 1,2 and 3) three times whenever it goes through the outer loop once. Therefore, the inner loop gets done 4 times 3, or 12 times. This allows the program to check all ordered pairs for the optimum profit value.

Statements 4 and 6 arrange to have x be 0, 1, 2, and 3, while y is changing from 0 to 1 to 2 as desired in our question.

Statement 7 evaluates the function in FORTRAN IV and assigns the current value to P.

Statement 8 checks to see if the current value of P is greater than the current value of PMAX. If it is, then it is a possible maximum value and it follows the command GO TO 2 and jumps to the storage space

```
IX=X
IY=Y
PMAX=P
```

which stores the current value of x,y and PMAX (possible optimum points). Then, it proceeds with the next loop. However, if P is not greater than the current value of PMAX statement 8 is false; therefore, the command GO TO 2 is ignored and the program goes to the next statement, GO TO 1, which skips over the storage space (statements 10 to 12) and continues with the next loop.

The result is that all 12 ordered pairs are checked to see if they produce a $P(x,y)$ value larger than any previous ones. If they do, the x and y and current maximum P value are stored in IX, IY and PMAX, respectively, as follows:

```
IX=X
IY=Y
PMAX=P
```

So at the end of the program, the current optimum solution (which is stored) is the true optimum because all points have been checked.

Statements 14 and 15 arrange to print the maximum solution.

Statements 16 and 17 just stop the program. These are used at the end of a FORTRAN IV program.

Now, if the conditions of the practical problem were such that x could take the values $0,1,2,\ldots,3999$, while y ranged through the values $0,1,2,\ldots,2999$, then we would have 12 million points to check instead of 12. This could be accomplished by changing DO 1 I=1,4 to DO 1 I=1,4000 and changing DO 1 J=1,3 to DO 1 J=1,3000. That's all that needs to be done.

The thoughtful reader may be wondering about this approach if we had, say, 20 variables and 100 possible values for each variable. Then we would have 100^{20} combinations of 20 variables to check. Even on the world's fastest computer this would take years. This is a valid question and the second part of the book will demonstrate how to get a solution in those cases. There are ways around that problem. Although this was not the case a few years ago, today if you can write the optimization problem down on paper, it should be a simple matter to optimize the function in question. This text will devote itself to that subject.