CS 91 USACO

Bronze Division

Unit 5: Number Algorithms

LECTURE 21: NUMBER PROGRAMMING PARADIGMS

DR. ERIC CHOU IEEE SENIOR MEMBER

Objectives

- Number programming paradigms
- Practice: Ski Design
- Practice: Arithmetic Progressions

Number-based Programming Paradigms

SECTION 2

Programming Paradigms

- 0-1 Programming
- Integer Programming
- Linear Programming
- Quadratic Programming
- Dynamic Programming
- Inductive versus deductive programming

Zero-One Integer Programming

- Zero-one integer programming (which can also be written as '0-1' integer programming) is a mathematical method of using a series of binary functions; in particular, yes ('1') and no ('0') answers to arrive at a solution when there are two mutually exclusive options.
- In the world of finance, zero-one integer programming is often used to provide answers to capital rationing problems, as well as to optimize investment returns and assist in planning, production, transportation, and other issues.

Zero-One Integer Programming

- Zero-one integer programming relies on mutually exclusive yes (1) and no (0) decisions to find solutions to logic problems.
- In zero-one integer problems, each variable is represented only by 0 ('no') or 1 ('yes'), and could represent selecting or rejecting an option, turning on or off electronic switches, or a straight-forward yes or no answer used in various other applications.
- This type of programming can be useful for companies making decisions on issues like what to invest in or which of two proposed products are easiest to manufacture.

X_i = 1 if project ***i*** is selected, 0 otherwise

Solving using LINDO or Excel's Solver

$X_1=1$ $X_2=0$ $X_3=1$ $X_4=1$

Optimal Return = 414

Maximize $\mathbf{217}X_1 + \mathbf{125}X_2 + \mathbf{88}X_3 + \mathbf{109}X_4$

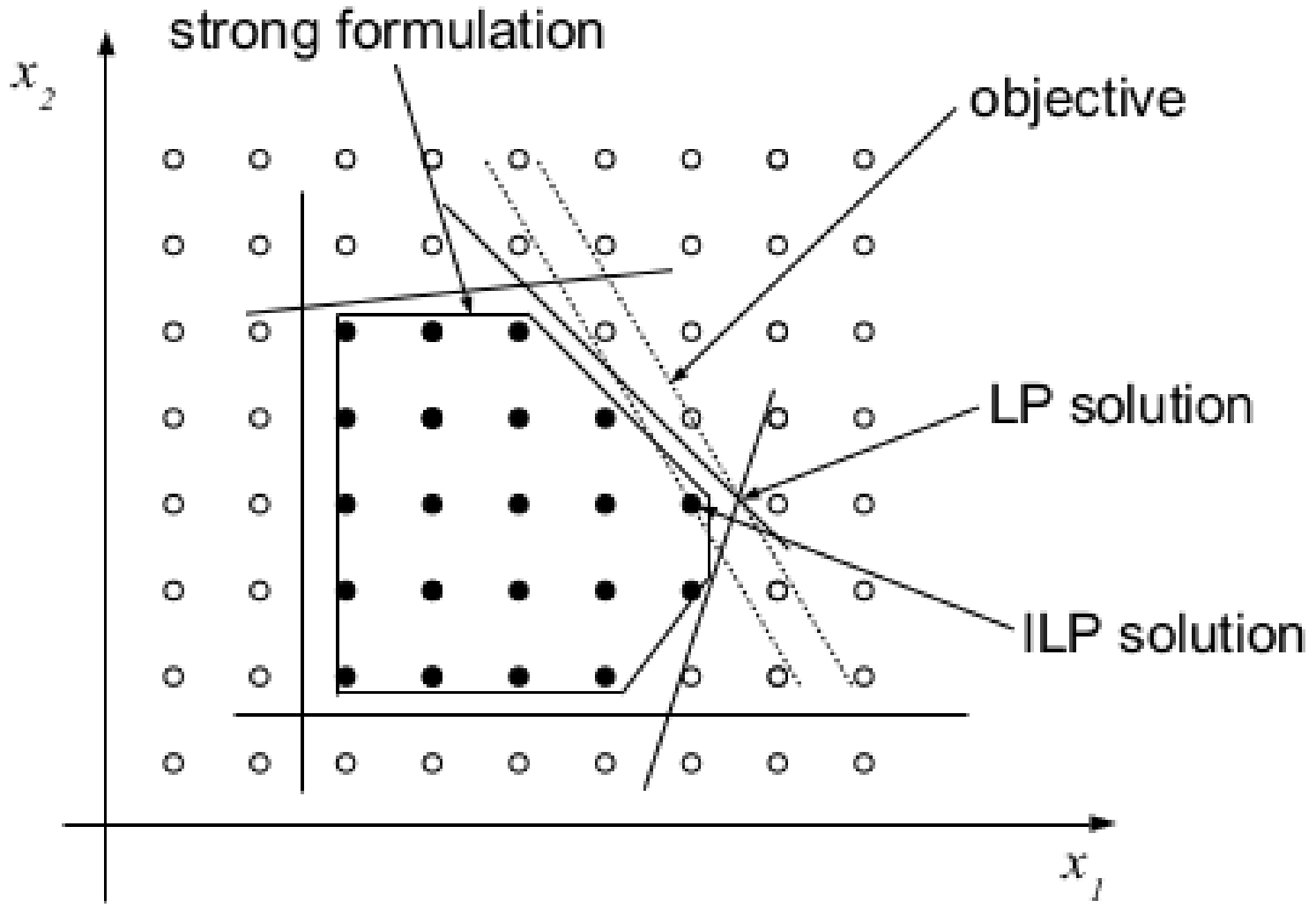
s.t.

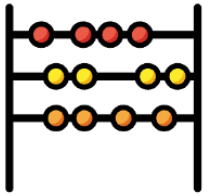
$\mathbf{58}X_1 + \mathbf{44}X_2 + \mathbf{26}X_3 + \mathbf{23}X_4 \leq \mathbf{120}$ *(January)*

$\mathbf{25}X_1 + \mathbf{29}X_2 + \mathbf{13}X_3 + \mathbf{17}X_4 \leq \mathbf{80}$ *(February)*

$\mathbf{43}X_1 + \mathbf{25}X_2 + \mathbf{23}X_3 + \mathbf{29}X_4 \leq \mathbf{95}$ *(March)*

X_i = 0-1


strong formulation
objective
LP solution
ILP solution
x_2
x_1



	Linear Programming	**Machine Learning**
Optimal solution	✓	✗
Inference time	*Slow*	*Fast*
Expert knowledge	*Required*	*Not required*
Training data	*Not required*	*Required*
Non-linear functions	✗	✓

Practice: Ski Design

SECTION 2

Problem Statement

- Farmer John has N hills on his farm (1 <= N <= 1,000), each with an integer elevation in the range 0 .. 100. In the winter, since there is abundant snow on these hills, FJ routinely operates a ski training camp.
- Unfortunately, FJ has just found out about a new tax that will be assessed next year on farms used as ski training camps. Upon careful reading of the law, however, he discovers that the official definition of a ski camp requires the difference between the highest and lowest hill on his property to be strictly larger than 17. Therefore, if he shortens his tallest hills and adds mass to increase the height of his shorter hills, FJ can avoid paying the tax as long as the new difference between the highest and lowest hill is at most 17.

Problem Statement

- If it costs x^2 units of money to change the height of a hill by x units, what is the minimum amount of money FJ will need to pay? FJ can change the height of a hill only once, so the total cost for each hill is the square of the difference between its original and final height. FJ is only willing to change the height of each hill by an integer amount.

INPUT FORMAT (skidesign.in):

Line 1: The integer N.
Lines 2..1+N: Each line contains the elevation of a single hill.

SAMPLE INPUT:

```
5
20
4
1
24
21
```

INPUT DETAILS:

FJ's farm has 5 hills, with elevations 1, 4, 20, 21, and 24.

OUTPUT FORMAT (skidesign.out):

- The minimum amount FJ needs to pay to modify the elevations of his hills so the difference between largest and smallest is at most 17 units.
- Line 1:

SAMPLE OUTPUT:

```
18
```

Output Details

FJ keeps the hills of heights 4, 20, and 21 as they are. He adds mass to the hill of height 1, bringing it to height 4 (cost = 3^2 = 9). He shortens the hill of height 24 to height 21, also at a cost of 3^2 = 9.

Nature of the Problem

- Complete search of the minimum cost function.
- The cost function is defined as the squared out_of_zone penalty for each data points.
- Find the maximum and minimum of the data sets.
- Traverse through i=min to i=max-17 cases. Find the case that will produce the minimum penalties.

Complete Search of the Solution Space

hill height	elevation intervals and cost							
	(0,17)	(1,18)	(2,19)	(3,20)	(4,21)	(5,22)	(6,23)	(7,24) ....
1	0	0	1	4	9	16	25	36
4	0	0	0	0	0	1	4	9
20	9	4	1	0	0	0	0	0
21	16	9	4	1	0	0	0	0
24	49	36	25	16	9	4	1	0
total	74	49	31	21	*18*	21	30	45

Practice: Arithmetic Progressions

SECTION 3

Problem Statement

- An arithmetic progression is a sequence of the form a, a+b, a+2b, ..., a+nb where n=0, 1, 2, 3, ... For this problem, a is a non-negative integer and b is a positive integer.
- Write a program that finds all arithmetic progressions of length n in the set S of bisquares. The set of bisquares is defined as the set of all integers of the form $p^2 + q^2$ (where p and q are non-negative integers).

INPUT FORMAT (ariprog.in):

- Line 1: N (3 <= N <= 25), the length of progressions for which to search
- Line 2: M (1 <= M <= 250), an upper bound to limit the search to the bisquares with 0 <= p,q <= M

SAMPLE INPUT:

```
5
7
```

OUTPUT FORMAT ariprog.out):

- If no sequence is found, a single line reading `NONE'. Otherwise, output one or more lines, each with two integers: the first element in a found sequence and the difference between consecutive elements in the same sequence. The lines should be ordered with smallest-difference sequences first and smallest starting number within those sequences first.
- There will be no more than 10,000 sequences.

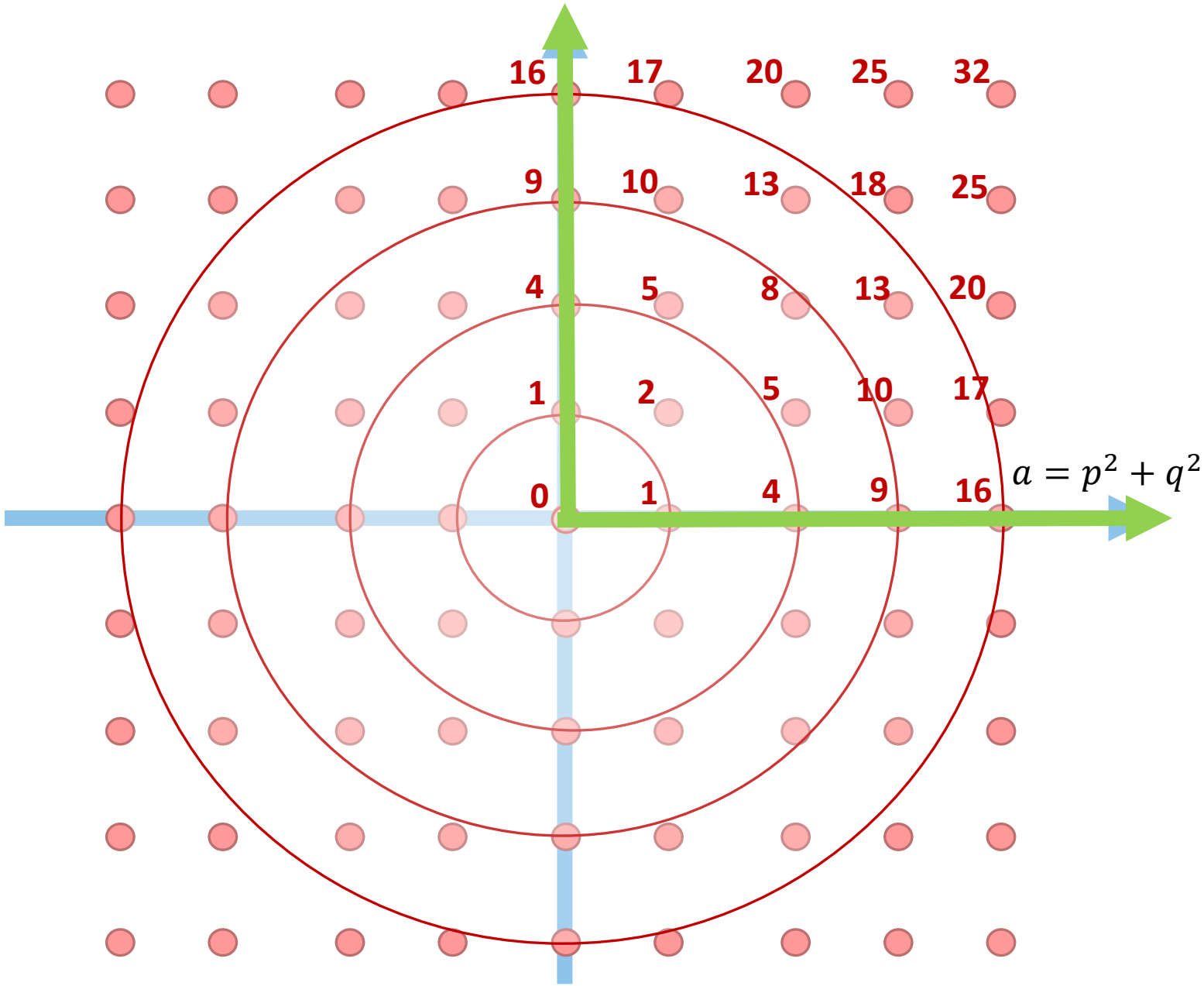
SAMPLE OUTPUT:

```
1 4
37 4
2 8
29 8
1 12
5 12
13 12
17 12
5 20
2 24
```

Bi-Square number

- A bi-square number $a = p^2 + q^2$. Where a is a number from 0 to $2M^2$, where p and q are number from 0 to M.
- And let's assign $sum = 2M^2$. And, the program will just check the test cases with $1 \leq M \leq 250$.

- How to check is a number is bi-square or not?
- Let $r = a - p^2$. If r is the square of a number q, then a is bi-square.


16
17
20
25
32
9
10
13
18
25
4
5
8
13
20
1
2
5
10
17
0
1
4
9
16
$a = p^2 + q^2$

Collect all the bi-square number given a specific M.

- If M = 4, maximum number for sum will be 2 * 4 * 4 = 32. Then, the bi-square number are
- [0, 1, 2, 4, 5, 8. 9. 10. 13. 16, 17, 18, 20, 25, 43]. These are shown as the points on the grid in the Figure in the previous page.
- The list of the bi-square numbers depends on M a lot.
- Therefore, for this problem, we should build a list of bi-square number. It can be represented as arraylist or an array for availability.
- The available array is of better performance.

hasSquare(n)

Evaluate whether n is a bi-square number or not.

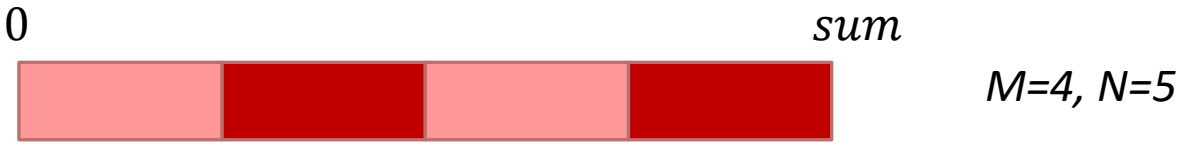
```
public static boolean hasBisquare(int n){
      for (int p=minM; p<=maxM; p++){
            int p2 = p*p;
            if (p2 > n) { break; }
            int r  = n - p2;
            int qlow  = (int) Math.floor(Math.sqrt(r));
            int qhigh = (int) Math.ceil(Math.sqrt(r));
            int q =0;

            if (qlow>=minM && qlow <=maxM ||  qlow>=minM && qlow <=maxM)
               if (r==qlow*qlow || r==qhigh*qhigh) { return true; }
       }
      return false;
   }
```

Nature of the problem

- Integer programming. Use only valid integers to minimize the execution time.
- Because the problem has the time constraint so the program performance is essential.
- Build the bi-square number list first.
- Check the sequences of length **N** with difference **d**.
- Maximum difference to be check for a given N is

$$d_{max} = \frac{2M^2}{(N-1)}$$



M=4, N=5

M=4, d=2

[0, 1, 2, 4, 5, 8, 9, 10, 13. 16, 17, 18, 20, 25, 43]

0 2
2 2
8 2
16 2
18 2

M=4, N=2

[0, 1, 2, 4, 5, 8, 9, 10, 13. 16, 17, 18, 20, 25, 43]

0 1	1 3	0 5	1 7	0 9	1 12	1 15	0 17	2 23
1 1	2 3	4 5	2 7	1 9	4 12	2 15	1 17	9 23
4 1	5 3	5 5	9 7	4 9	5 12	5 15	8 17	1 24
8 1	10 3	8 5	10 7	8 9	8 12	10 15	0 18	8 24
9 1	13 3	13 5	13 7	9 9	13 12	17 15	2 18	0 25
16 1	17 3	20 5	18 7	16 9	20 12	0 16	1 19	5 27
17 1	0 4	2 6	25 7	0 10	0 13	1 16	13 19	4 28
0 2	1 4	4 6	0 8	8 10	4 13	2 16	0 20	2 30
2 2	4 4	10 6	1 8	10 10	5 13	4 16	5 20	1 31
8 2	5 4		2 8	2 11	2 14	9 16	4 21	0 32
16 2	9 4		5 8	5 11	4 14	16 16	10 22	
18 2	13 4		8 8	9 11	18 14			
	16 4		9 8					
			10 8					
			17 8					

Nature of the problem

- Integer programming. Use only valid integers to minimize the execution time.
- Because the problem has the time constraint so the program performance is essential.
- Build the bi-square number list first.
- Check the sequences of length **N** with difference **d**.
- Check the combination of a_0 and d, if all a_0 to a_{N-1} are all bi-square number, then, report a_0 and d