Arithmetic Sequence Puzzles

Valentina Wu – up201907483
Victor Saldanha Nunes – up201907226

Problem Presentation

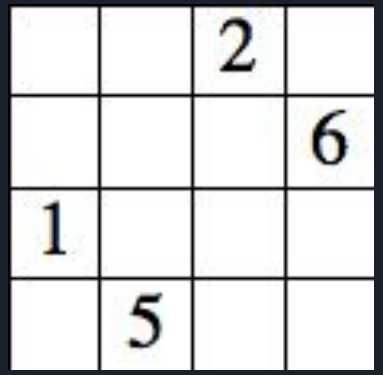
Fill in some of the blank squares with digits 0-9.
Each row and column should contains exactly 3 digits, and these digits should form an increasing arithmetic sequence, a sequence with a common difference.

		2	
			6
1			
	5		

	1	2	3
0	3		6
1		5	9
2	5	8	

Model of input and output

The input and output are modeled as a list of lists. The -1 element represents an empty square, and the other numbers represent a square filled with that number.




```
[[-1, -1, 2, -1],
 [-1, -1, -1, 6],
 [1, -1, -1, -1],
 [-1, 5, -1, -1]]
```

Prolog Solution

Prolog - Problem Solution Overview

```
%solve(+InputBoard, -OutputBoard)
solve(InputBoard, OutputBoard) :- statistics(walltime, [Start,_]),
                                  length(InputBoard, BoardSize), length(OutputBoard, BoardSize),
                                  length(PositionsHBoard, BoardSize), length(PositionsVBoard, BoardSize),
                                  checkHorizontal(InputBoard, OutputBoard, PositionsHBoard),
                                  checkVertical(InputBoard, OutputBoard, PositionsVBoard),
                                  append(PositionsHBoard, PlainPositionsHBoard),
                                  append(PositionsVBoard, PlainPositionsVBoard),
                                  append(PlainPositionsVBoard, PlainPositionsHBoard, Vars),
                                  labeling([dom_w_deg, median], Vars),
                                  statistics(walltime, [End,_]), Time is End - Start,
                                  format('Time spent to find the answer: ~3d s~n', [Time]).
```

Prolog - solving each line and column

```
%checkHorizontal(+InputBoard, -OutputBoard)
checkHorizontal([],[],[]).
checkHorizontal([InputLine | InputRemainingLines], [OutputLine | OutputRemainingLines],
                [PositionsLine | PositionsRemainingLine]) :- solveLine(InputLine, OutputLine, PositionsLine),
                                                             checkHorizontal(InputRemainingLines, OutputRemainingLines, PositionsRemainingLine).

%solveLine(+InputLine, -OutputLine)
solveLine(InputLine, OutputLine, [N1, N1Position, N2, N2Position, N3, N3Position]) :- length(InputLine, LineSize), length(OutputLine, LineSize),
                                findLineNumber(InputLine, N3, N3Position),
                                element(N3Position, OutputLine, N3),
                                domain(OutputLine, -1, 9),
                                domain([N1, N2], 0, 9), all_distinct([N1, N2, N3]),
                                domain([N1Position, N2Position], 1, LineSize), all_distinct([N1Position, N2Position, N3Position]),
                                constrainNumbers([N1, N2, N3], [N1Position, N2Position, N3Position]),
                                element(N1Position, OutputLine, N1), element(N2Position, OutputLine, N2),
                                NumberOfEmptySquares is LineSize - 3,
                                exactly(-1, OutputLine, NumberOfEmptySquares).
```

```
checkVertical(InputBoard, OutputBoard, PositionsVBoard) :- transpose(InputBoard, InputBoardTransposed), transpose(OutputBoard, OutputBoardTransposed),
                                checkHorizontal(InputBoardTransposed, OutputBoardTransposed, PositionsVBoard).
```

Prolog - constraining N1, N2 and N3

```
%findLineNumber(+InputLine, -N3, -N3Position)
findLineNumber(InputLine, N3, N3Position) :- element(N3Position, InputLine, N3), N3 #\= -1.

%constrainNumbers(+[N1, N2, N3], +[N1Position, N2Position, N3Position])
constrainNumbers([N1, N2, N3], [N1Position, N2Position, N3Position]) :- C in 1..4,
                                        (N3 #= N2 + C #/\ N2 #= N1 + C #/\ N1Position #< N2Position #/\ N2Position #< N3Position) +
                                        (N2 #= N3 + C #/\ N3 #= N1 + C #/\ N1Position #< N3Position #/\ N3Position #< N2Position) +
                                        (N3 #= N1 + C #/\ N1 #= N2 + C #/\ N2Position #< N1Position #/\ N1Position #< N3Position) +
                                        (N1 #= N3 + C #/\ N3 #= N2 + C #/\ N2Position #< N3Position #/\ N3Position #< N1Position) +
                                        (N2 #= N1 + C #/\ N1 #= N3 + C #/\ N3Position #< N1Position #/\ N1Position #< N2Position) +
                                        (N1 #= N2 + C #/\ N2 #= N3 + C #/\ N3Position #< N2Position #/\ N2Position #< N1Position) #= 1.
```

Experimental Setup - Variable Ordering

Variable Ordering (time/s)				
Method	4x4	5x5	6x6	7x7
leftmost	0,014s	1,310s	48,957s	X
min	0,066s	63,881s	X	X
max	2,001s	X	X	X
ff	0,012s	0,614s	1,208s	294,055s
anti_first_fail	5,536s	X	X	X
occurrence	0,015s	11,993s	23,983s	X
ffc	0,014s	0,296s	3,087s	12,881s
max_regret	0,005s	1,542s	10,243s	X
impact	0,004s	9,522s	X	X
dom_w_deg	**0,013s**	**0,056s**	**0,644s**	**7,083s**

Experimental Setup - Variable Selection

Variable Selection (time/s)

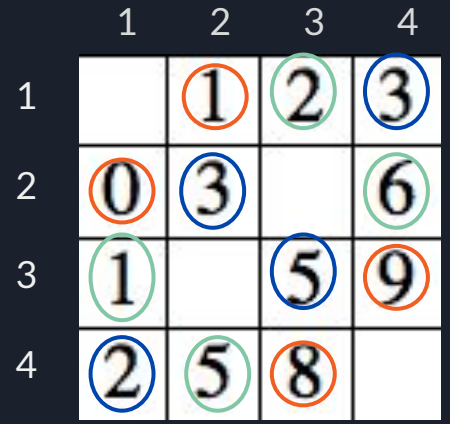
Method	6x6	7x7
ffc step	0,579s	20,104s
ffc enum	0,568s	19,784s
ffc bisect	0,578s	20,096s
ffc median	0,134s	89,664s
ffc middle	0,267s	58,175s

Variable Selection (time/s)

Method	6x6	7x7
dom_w_deg step	0,182s	19,011s
dom_w_deg enum	0,552s	42,620s
dom_w_deg bisect	0,164s	17,115s
dom_w_deg median	**0,644s**	**15,452s**
dom_w_deg middle	5,308s	98,907s

CPlex Solution

CPLEX - Variables




```
using CP;

int n = ...; //Matrix nxn
int InputBoard[1..n][1..n] = ...;

range Rows = 1..n;
range Cols = 1..n;
range Numbers = 0..9;

dvar int OutputBoard[1..n][1..n] in -1..9;
dvar int N1_Rows[Rows] in Numbers;
dvar int N2_Rows[Rows] in Numbers;
dvar int N3_Rows[Rows] in Numbers;
dvar int N1_Cols[Cols] in Numbers;
dvar int N2_Cols[Cols] in Numbers;
dvar int N3_Cols[Cols] in Numbers;
dvar int N1Position_Rows[Rows] in Cols;
dvar int N2Position_Rows[Rows] in Cols;
dvar int N3Position_Rows[Rows] in Cols;
dvar int N1Position_Cols[Cols] in Rows;
dvar int N2Position_Cols[Cols] in Rows;
dvar int N3Position_Cols[Cols] in Rows;
```

CPLEX - Constraints

```
38  forall (r in Rows, c in Cols){
39    //findLineNumber(InputLine, N3, N3Position),
40    (InputBoard[r][c] != -1) => (N3_Rows[r]==InputBoard[r][c] && N3Position_Rows[r] == c) &&
41                (N3_Cols[c]==InputBoard[r][c] && N3Position_Cols[c] == r);
42
43    /* element(N1Position, OutputLine, N1)
44    element(N2Position, OutputLine, N2)
45    element(N3Position, OutputLine, N3)*/
46    N1_Rows[r] == OutputBoard[r][N1Position_Rows[r]];
47    N2_Rows[r] == OutputBoard[r][N2Position_Rows[r]];
48    N3_Rows[r] == OutputBoard[r][N3Position_Rows[r]];
49
50    N1_Cols[c] == OutputBoard[N1Position_Cols[c]][c];
51    N2_Cols[c] == OutputBoard[N2Position_Cols[c]][c];
52    N3_Cols[c] == OutputBoard[N3Position_Cols[c]][c];
53  }
```

CPLEX - Constraints

```
64    //constrainNumbers(+[N1, N2, N3], +[N1Position, N2Position, N3Position])
65    forall(j in Cols){
66      // Rows
67      (N1_Rows[j] < N2_Rows[j] && N2_Rows[j] < N3_Rows[j]) => N2_Rows[j]-N1_Rows[j]==N3_Rows[j]-N2_Rows[j] &&
68         N1Position_Rows[j] < N2Position_Rows[j] && N2Position_Rows[j] < N3Position_Rows[j];
69      (N1_Rows[j] < N3_Rows[j] && N3_Rows[j] < N2_Rows[j]) => N3_Rows[j]-N1_Rows[j]==N2_Rows[j]-N3_Rows[j] &&
70         N1Position_Rows[j] < N3Position_Rows[j] && N3Position_Rows[j] < N2Position_Rows[j];
71      (N2_Rows[j] < N1_Rows[j] && N1_Rows[j] < N3_Rows[j]) => N1_Rows[j]-N2_Rows[j]==N3_Rows[j]-N1_Rows[j] &&
72         N2Position_Rows[j] < N1Position_Rows[j] && N1Position_Rows[j] < N3Position_Rows[j];
73      (N2_Rows[j] < N3_Rows[j] && N3_Rows[j] < N1_Rows[j]) => N3_Rows[j]-N2_Rows[j]==N1_Rows[j]-N3_Rows[j] &&
74         N2Position_Rows[j] < N3Position_Rows[j] && N3Position_Rows[j] < N1Position_Rows[j];
75      (N3_Rows[j] < N1_Rows[j] && N1_Rows[j] < N2_Rows[j]) => N1_Rows[j]-N3_Rows[j]==N2_Rows[j]-N1_Rows[j] &&
76         N3Position_Rows[j] < N1Position_Rows[j] && N1Position_Rows[j] < N2Position_Rows[j];
77      (N3_Rows[j] < N2_Rows[j] && N2_Rows[j] < N1_Rows[j]) => N2_Rows[j]-N3_Rows[j]==N1_Rows[j]-N2_Rows[j] &&
78         N3Position_Rows[j] < N2Position_Rows[j] && N2Position_Rows[j] < N1Position_Rows[j];
79
80      //Cols
81      (N1_Cols[j] < N2_Cols[j] && N2_Cols[j] < N3_Cols[j]) => N2_Cols[j]-N1_Cols[j]==N3_Cols[j]-N2_Cols[j] &&
82         N1Position_Cols[j] < N2Position_Cols[j] && N2Position_Cols[j] < N3Position_Cols[j];
83      (N1_Cols[j] < N3_Cols[j] && N3_Cols[j] < N2_Cols[j]) => N3_Cols[j]-N1_Cols[j]==N2_Cols[j]-N3_Cols[j] &&
84         N1Position_Cols[j] < N3Position_Cols[j] && N3Position_Cols[j] < N2Position_Cols[j];
85      (N2_Cols[j] < N1_Cols[j] && N1_Cols[j] < N3_Cols[j]) => N1_Cols[j]-N2_Cols[j]==N3_Cols[j]-N1_Cols[j] &&
86         N2Position_Cols[j] < N1Position_Cols[j] && N1Position_Cols[j] < N3Position_Cols[j];
87      (N2_Cols[j] < N3_Cols[j] && N3_Cols[j] < N1_Cols[j]) => N3_Cols[j]-N2_Cols[j]==N1_Cols[j]-N3_Cols[j] &&
88         N2Position_Cols[j] < N3Position_Cols[j] && N3Position_Cols[j] < N1Position_Cols[j];
89      (N3_Cols[j] < N1_Cols[j] && N1_Cols[j] < N2_Cols[j]) => N1_Cols[j]-N3_Cols[j]==N2_Cols[j]-N1_Cols[j] &&
90         N3Position_Cols[j] < N1Position_Cols[j] && N1Position_Cols[j] < N2Position_Cols[j];
91      (N3_Cols[j] < N2_Cols[j] && N2_Cols[j] < N1_Cols[j]) => N2_Cols[j]-N3_Cols[j]==N1_Cols[j]-N2_Cols[j] &&
92         N3Position_Cols[j] < N2Position_Cols[j] && N2Position_Cols[j] < N1Position_Cols[j];
93    }
```

CPLEX - Constraints

```
55  /*all_distinct([N1, N2, N3])
56   all_distinct([N1Position, N2Position, N3Position])*/
57  forall(i in Rows){
58    N1_Rows[i] != N2_Rows[i] && N2_Rows[i] != N3_Rows[i] && N1_Rows[i] != N3_Rows[i];
59    N1_Cols[i] != N2_Cols[i] && N2_Cols[i] != N3_Cols[i] && N1_Cols[i] != N3_Cols[i];
60    N1Position_Rows[i] != N2Position_Rows[i] && N2Position_Rows[i] != N3Position_Rows[i] && N1Position_Rows[i] != N3Position_Rows[i];
61    N1Position_Cols[i] != N2Position_Cols[i] && N2Position_Cols[i] != N3Position_Cols[i] && N1Position_Cols[i] != N3Position_Cols[i];
62  }
```

```
96   forall(r in Rows, c in Cols){
97
98        (N1Position_Rows[r] == c) => OutputBoard[r][c] == N1_Rows[r];
99        (N2Position_Rows[r] == c) => OutputBoard[r][c] == N2_Rows[r];
100       (N3Position_Rows[r] == c) => OutputBoard[r][c] == N3_Rows[r];
101       (N1Position_Rows[r] != c && N2Position_Rows[r] != c && N3Position_Rows[r] != c) => OutputBoard[r][c] == -1;
102
103       (N1Position_Cols[c] == r) => OutputBoard[r][c] == N1_Cols[c];
104       (N2Position_Cols[c] == r) => OutputBoard[r][c] == N2_Cols[c];
105       (N3Position_Cols[c] == r) => OutputBoard[r][c] == N3_Cols[c];
106       (N1Position_Cols[c] != r && N2Position_Cols[c] != r && N3Position_Cols[c] != r) => OutputBoard[r][c] == -1;
107
108   }
```

CPLEX - Variable and Value Selection

```
112 execute{
113   var f = cp.factory;
114   var phase1 = f.searchPhase(OutputBoard,f.selectLargest(f.regretOnMin()),f.selectSmallest(f.valueImpact()));
115   cp.setSearchPhases(phase1);
116   cp.param.SearchType = "IterativeDiving";
117 }
```

Experimental Setup - Search Types

SEARCH TYPES (time/s)		
	4x4	5x5
None	0.08s	0.35s
Depth-first	0.06s	0.38s
Restart	0.11s	0.32s
Multi-point	0.09s	0.42s
Iterative-diving	**0.06s**	**0.33s**

Experimental Setup - Variable Selection

VARIABLE SELECTION (time/s)					
selectSmallest	4x4	5x5	selectLargest	4x4	5x5
domainSize()	0.07s	0.33s	domainSize()	0.07s	0.38s
domainMin()	0.14s	0.38s	domainMin()	0.09s	0.34s
domainMax()	0.10s	0.32s	domainMax()	0.10s	0.30s
regretOnMin()	0.11s	0.40s	regretOnMin()	**0.09s**	**0.29s** 2
regretOnMax()	0.06s	0.36s	regretOnMax()	0.09s	0.39s
successRate()	0.07s	0.33s	successRate()	0.10s	0.30s
impact()	0.06s	0.38s	impact()	0.08s	0.39s
localImpact()	0.08s	0.35s	localImpact()	0.09s	0.33s
impactOfLast Branch()	**0.06s**	**0.33s** 1	impactOfLast Branch()	0.09s	0.39s

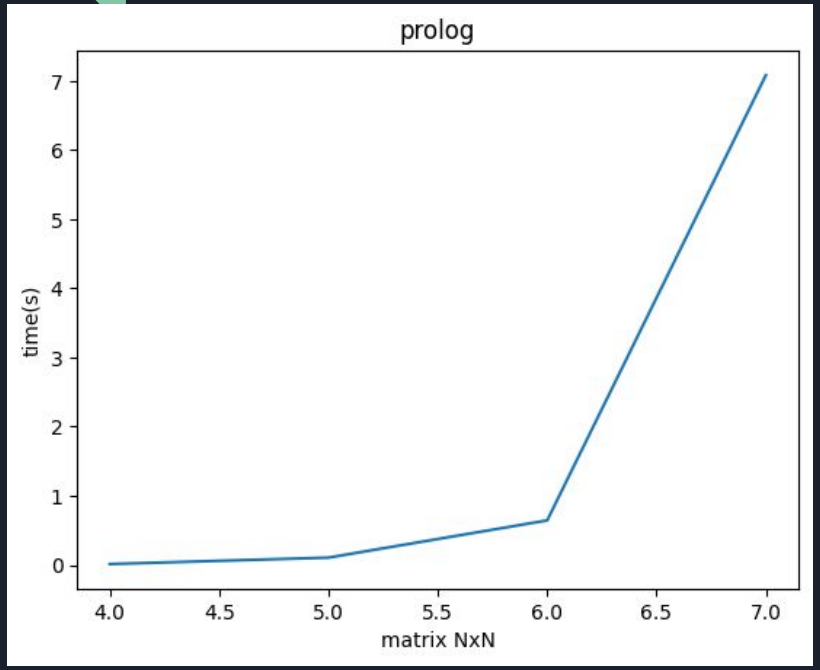
Experimental Setup - Value Selection

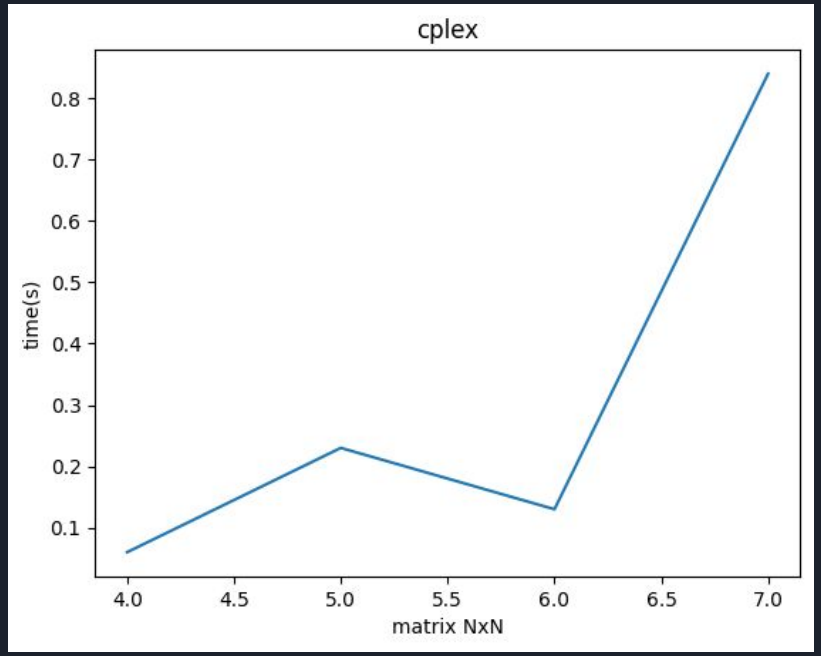
VALUE SELECTION (time/s)						
selectSmallest	4x4	5x5		selectLargest	4x4	5x5
value()	0.10s	0.30s		value()	0.09s	0.32s
valueImpact()	**0.10s**	**0.29s**	3	valueImpact()	0.09s	0.37s
valueSuccess Rate()	0.07s	0.36s		valueSuccess Rate()	**0.09s**	**0.30s** 4

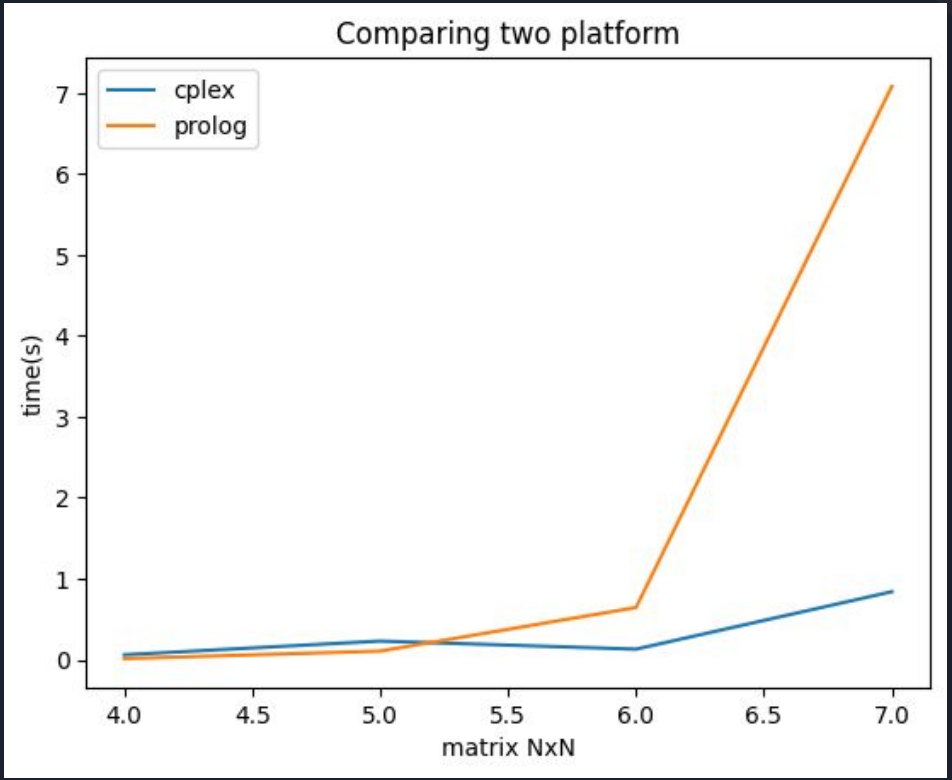
	4x4	5x5
2 & 4	0.10s	0.38s
2 & 3	**0.08s**	**0.37s**
1 & 3	0.09s	0.36s
1 & 4	0.10s	0.39s

```
var f = cp.factory;
var phase1 = f.searchPhase(OutputBoard,
     f.selectLargest(f.regretOnMin()),f.selectSmallest(f.valueImpact()));
cp.setSearchPhases(phase1);
```

Results


prolog
time(s)
matrix NxN
0
1
2
3
4
5
6
7
4.0
4.5
5.0
5.5
6.0
6.5
7.0


cplex
time(s)
matrix NxN
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
4.0
4.5
5.0
5.5
6.0
6.5
7.0


Comparing two platform
cplex
prolog
time(s)
0
1
2
3
4
5
6
7
4.0
4.5
5.0
5.5
6.0
6.5
7.0
matrix NxN

Thank You