

The background of the slide is a 3D rendering of a large puzzle. Most of the puzzle pieces are light gray, but a large section in the center is a solid blue. On top of this blue section, there is a dark purple silhouette of a person standing with their arms raised in a 'V' shape. The text is overlaid on the blue puzzle section.

PROJECT: 3-IN-A-ROW & ABC PATH (TEAM 5)

ISEN – 620 Survey Optimization

Client Name:

Prof. Erick Moreno-Centeno

Team Members:

- Hamza Zaim
- Mayank Jaggi
- Vedant Mehta

Phase III Report

Executive Summary

The aim of this project is to solve 3-In-A-Row and ABC path puzzle for the given problem statement by creating a solver. 3-In-A-Row puzzle requires the solver to fill the grey blocks in the cells with either blue or white cells whilst there are no more than two consecutive cells along rows and columns of same color. ABC path puzzle requires the solver to fill alphabets A to Y in a grid of 5x5 as per the hints provided along rows, columns and diagonals of the grid. The position of A in the grid is provided along with the puzzle.

We came up with two Binary Integer Linear Programming Models (one for each puzzle) which are flexible enough to support any initial version of the two puzzles, i.e. a square grid of any even number of rows and columns for 3-In-A-Row puzzle and a 5x5 grid with any combination of hints for 24 alphabets and initial position for alphabet A.

The parameters are defined to be fed into the model in a separate data file. Linear constraints are defined for the two puzzles as per their rules and requirements. Model is generalized as much as possible to make it scalable.

The model (parameters, objective function and constraints) is coded as per AMPL syntax to solve the two IP formulations, giving binary matrices depicting the desired results.

In this context, a Python application was developed simplifying the user input and the model interpretability. A user manual to enlighten the client throughout the steps to be followed in order to solve each of the puzzles

Contents

Introduction	4
Puzzle 1: 3-In-A-Row.....	7
Overview:	7
How to solve 3-In-A-Row:	8
IP Model.....	10
Detailed Description of the Model:.....	12
AMPL Model:	14
Problem Statement:	15
AMPL Data:	16
How to insert the data in the AMPL data file?	17
Puzzle 2: ABC Path	20
Overview:	20
How to solve ABC Path:	21
IP Model:.....	22
Detailed Description of the Model:.....	25
AMPL Model:	29
AMPL Data:	31
How to insert the data in the AMPL data file?	32
Technical Explanation	38
User Manual	39
3-In-A-Row Puzzle:	39
ABC Path Puzzle:.....	43
Conclusion	46
References.....	47
Appendix	48

Introduction

Linear Programming is a branch of Operations Research which helps you take informed decisions based on the information one has regarding the resources/services/constraints in the system. Linear Programming usually helps take decisions in order to maximize profit or minimize cost (basically making efficient decisions).

Real world problems are very complex and has a lot of constraints/restrictions involved which might be again dependent on other constraints. One uses linear programming to depict the complex relationships in a linear form. (Introductory Guide on Linear Programming, n.d.)

Linear Programming is used in day-to-day life without one knowing it. For example, taking the shortest path to a destination based on the time and traffic conditions, deciding strategies for ordering material, building production plan, etc. in a manufacturing environment, deciding number of people/commodities required based on requirements of the customers in a service environment, etc. are classic use cases of linear programming.

Integer Linear Programming is a special case of Linear Programming where decision variables can only take integer values. Integer Linear Programming is even more powerful tool as it very closely mirrors the real life conditions (where you can only hire whole number of people, or only produce whole number of products) and fractions can sometimes cause huge differences in profits or costs.

Integer programming can not only be used for solving profit maximization or cost minimization problems. It can also be used to model the mathematical puzzles, sometimes using binary variables (special case of Integer Programming problems), to solve large, complex puzzles like Sudoku, Nonogram, ABC Path, 3 in a row, etc. (Kira Goldner)

This report explores the application of Integer Programming to solve ABC Path and 3 in a row puzzle. This report also sheds some light on how to use AMPL to solve Optimization problems and also explains the AMPL code to solve the aforementioned puzzles.

Introduction to AMPL

In this phase of the project, the previously defined models must be translated into an AMPL files in order to solve any 3 in a row and ABC path puzzles.

A detailed description of the steps that should be followed in order to solve the puzzles, as well as the interpretation of the results for a better understanding of the solutions can also be found. (Robert Fourer)

What is AMPL?

A Mathematical Programming Language (AMPL) is an algebraic modeling language to describe and solve high-complexity problems for large-scale mathematical computing (i.e., large-scale optimization and scheduling-type problems). It was developed by Robert Fourer, David Gay, and Brian Kernighan at Bell Laboratories. AMPL supports dozens of solvers, both open source and commercial software (i.e. CPLEX). Problems are passed to solvers as “*.txt” files. AMPL is used by more than 100 corporate clients, and by government agencies and academic institutions. (Robert Fourer) (Wikipedia, n.d.)

Why AMPL?

The AMPL system supports the entire optimization modeling lifecycle — formulation, testing, deployment, and maintenance — in an integrated way that promotes rapid development and reliable results. Using a high-level algebraic representation that describes optimization models in the same ways that people think about them, AMPL can provide the head start you need to successfully implement large-scale optimization projects.

AMPL integrates its modeling language with a command language for analysis and debugging, and a scripting language for manipulating data and implementing optimization strategies. All use the same concepts to promote streamlined model-building. (Robert Fourer) (Wikipedia, n.d.)

Where can I find AMPL?

The software can be downloaded and simply installed via the company’s official website: <https://ampl.com/products/solvers/open-source/>

How to solve the puzzles using AMPL?

The following instructions must be respected in the stated order:

1. Download the AMPL software package from the website in the aforementioned URL
2. Unzip the downloaded package and extract all the files in a normal AMPL folder.
3. Double-click "sw.exe" in the created AMPL folder.
4. In the window that appears, type `ampl` and hit "Enter"
5. Then type `option solver cplex;` to select CPLEX as your solver
6. Input the model files:
 - For 3 in a row puzzle: `model p2t05mod3.txt;`
 - For ABC path puzzle: `model p2t05modA.txt;`
7. Input the data file: "data omcdat.txt";
 - For 3 in a row puzzle: `data p2t05dat3.txt;`
 - For ABC path puzzle: `data p2t05datA.txt;` (Moreno-Centeno, Instructions on AMPL and CPLEX)

Puzzle 1: 3-In-A-Row

Overview:

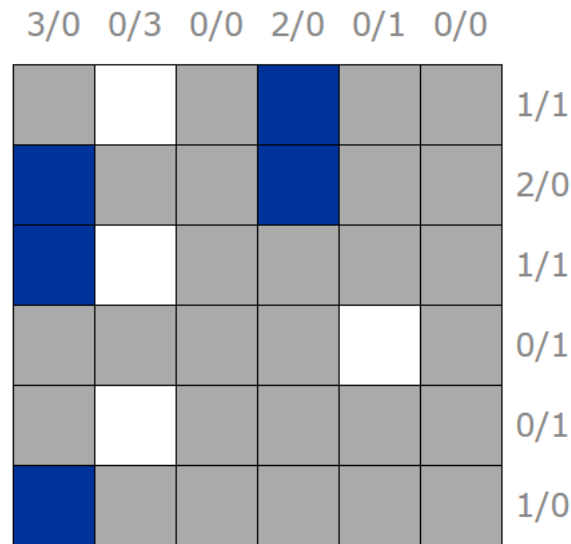


Figure 1 - 3-IN-A-ROW puzzle layout

- Fill the grid with blue and white squares
- No row or column should contain same colors three in a row
- Each of the row or column should contain **equal number** of blues and whites

Figure 1 shows the layout of the puzzle. Each puzzle has a unique solution. The next section will explain the steps of how to solve a puzzle (just a few initial steps) in order to better understand the thought process behind the puzzle. (Phase I Report)

How to solve 3-In-A-Row:

There will be a few cells in the grid that will be pre-filled with either blue or white and you can use those cells as the starting point

Let's start solving the puzzle:

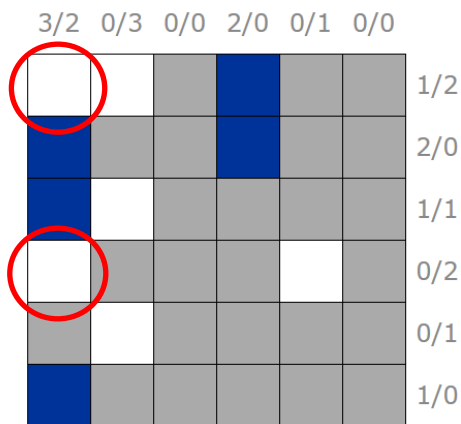


Figure 2 - Highlighted filled cells

Step 2 (Figure 3): Fill the highlighted cell with white because each row/column should have equal number of blues and whites

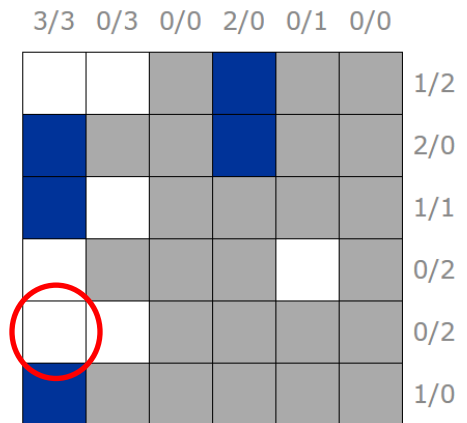


Figure 3 - Highlighted filled cell

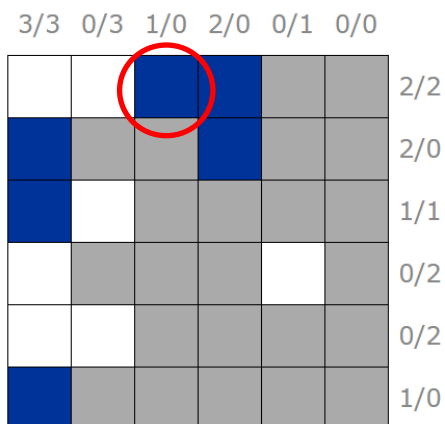


Figure 4 - Highlighted filled cell

Step 3 (Figure 4): Fill the highlighted cell with blue since we already have 2 consecutive whites in a row before the highlighted cell

Continue the same pattern until all the grey cells have been filled up with blue/white color. Figure 5 (below) shows the solved puzzle depicting all the blue and white cells. (Green color depicts the puzzle is correct)

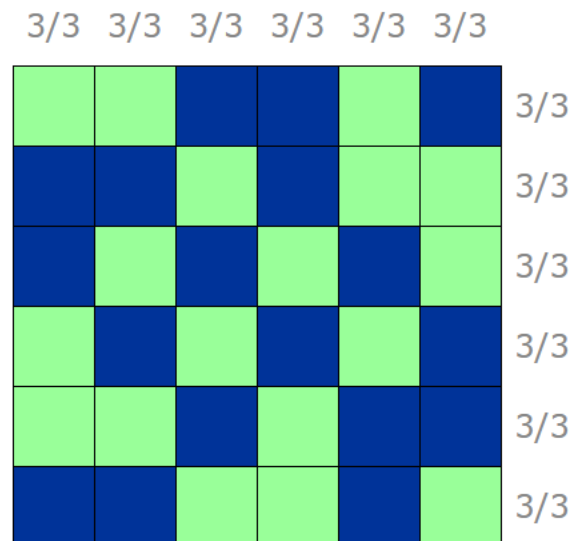


Figure 5 - Solved puzzle

Now that we have understood how to solve the puzzle, let's create an integer programming model to automate the puzzle solver. Simply put, we have used integer programming as a method to break down the "mathematical" value of each cell to be either 1 or 0 (based on color). This will become clearer as we start laying out the model and explain it. (Daily 3-In-A-Row, n.d.) (Phase I Report)

IP Model

Parameters:

S = Number of rows/columns, where S is an even number.

n = Number of invalid consecutive cells in a row/column, with $n \leq S$

$blue_{i,j}$ = Initial blue cells, for $i=1$ to S and $j=1$ to S .

$white_{i,j}$ = Initial white cells, for $i=1$ to S and $j=1$ to S .

Decision variables:

$$y_{i,j} = \begin{cases} 1 & \text{when cell } [i,j] \text{ is a blue cell} \\ 0 & \text{when cell } [i,j] \text{ is a white cell} \end{cases}, \text{ for } i=1 \text{ to } S \text{ and } j=1 \text{ to } S.$$

Objective Function:

Maximize colored_cells: 0

Constraints:

No more than $n-1$ consecutive blue cells in each row:

$$\sum_{k=0}^{n-1} y_{i,j+k} \leq n - 1, \text{ for } i=1 \text{ to } S, j=1 \text{ to } (S-(n-1))$$

No more than $n-1$ consecutive blue cells in each column:

$$\sum_{k=0}^{n-1} y_{i+k,j} \leq n - 1, \text{ for } i=1 \text{ to } (S-(n-1)), j=1 \text{ to } S$$

No more than $n-1$ consecutive white cells in each row:

$$\sum_{k=0}^{n-1} y_{i,j+k} \geq 1, \text{ for } i=1 \text{ to } S, j=1 \text{ to } (S-(n-1))$$

No more than $n-1$ consecutive white cells in each column:

$$\sum_{k=0}^{n-1} y_{i+k,j} \geq 1, \text{ for } i=1 \text{ to } (S-(n-1)), j=1 \text{ to } S$$

Each row has equal number of blue and white cells:

$$\sum_{j=1}^S y_{i,j} = \frac{S}{2}, \text{ for } i=1 \text{ to } S$$

Each column has equal number of blue and white cells:

$$\sum_{i=1}^S y_{i,j} = \frac{S}{2}, \text{ for } j=1 \text{ to } S$$

Prefilled blue cells:

$$blue_{i,j} \leq y_{i,j}, \text{ for } i=1 \text{ to } S, j=1 \text{ to } S$$

Prefilled white cells:

$$white_{i,j} \leq (1 - y_{i,j}), \text{ for } i=1 \text{ to } S, j=1 \text{ to } S$$

Binary requirement:

$$y_{i,j} \text{ is binary}, \text{ for } i=1 \text{ to } S, j=1 \text{ to } S \text{ (Phase I Report) (Moreno-Centeno, Phase I Model)}$$

Detailed Description of the Model:

Parameters:

The following parameters are defined for this model:

- Parameter S (square matrix)
- Parameter n (invalid consecutive cells)
- Parameter $blue_{i,j}$ (a matrix of size $S \times S$ with 1 where blue otherwise 0)
- Parameter $white_{i,j}$ (a matrix of size $S \times S$ with 1 where blue otherwise 0)

These parameters will change for each new puzzle and would have to be assigned values manually in the specified format in order to be fed into the model file.

Now, let's examine the model file and try to understand the objective function, decision variables, and its constraints.

Decision Variables:

For decision variables' purposes, index i depicts row number and index j depicts column number.

The decision variable $y_{i,j}$ is a binary variable, it has been selected in order to assign 1 to blue cells and 0 to white cells.

Objective Function:

This LP is not related to a maximization of profit or a minimization of cost. Thus, the 0 defined in our function simply refers to the fact that the model is to be solved based only on the defined constraints.

The following constraints make our algorithm follow the rules of the puzzle:

Constraints:

- **Constraint 1:** $\sum_{k=0}^{n-1} y_{i,j+k} \leq n - 1$, for $i=1$ to S , $j=1$ to $(S-(n-1))$

This constraint enumerates over all the values of i and j , and makes sure there are no more than n consecutive blue cells in each row (since we are jumping the index of j to $j+1$ to $j+2$ and the summation of these three values is less than two implying, at most, either of the two cells can be blue).

- **Constraint 2:** $\sum_{k=0}^{n-1} y_{i+k,j} \leq n - 1$, for $i=1$ to $(S-(n-1))$, $j=1$ to S

This constraint is very similar to the above constraint, the only difference being this constraint makes sure there are no more than n consecutive blue cells in each column.

- **Constraint 3:** $\sum_{k=0}^{n-1} y_{i,j+k} \geq 1$, for $i=1$ to S , $j=1$ to $(S-(n-1))$

This constraint makes sure there are no more than $n-1$ consecutive white cells in each row (and enumerates over all the values of i and j).

- **Constraint 4:** $\sum_{k=0}^{n-1} y_{i+k,j} \geq 1$, for $i=1$ to $(S-(n-1))$, $j=1$ to S

This constraint ensures there are no more than $n-1$ consecutive white cells in each column.

Now, the only thing that remains is to make sure we have only one color in each cell. None of the constraints above takes care of that.

- **Constraint 5:** $\sum_{j=1}^S y_{i,j} = \frac{S}{2}$, for $i=1$ to S

This constraint makes sure each row has equal number of blue and white cells (and we are enumerating over all the rows). Thus, sum of all the blues (each depicting value 1) would be equal to sum of all the whites (each depicting value 1) in each row.

- **Constraint 6:** $\sum_{i=1}^S y_{i,j} = \frac{S}{2}$, for $j=1$ to S

This constraint ensures that each column has equal number of blue and white cells (and we are enumerating over all the columns). Thus, sum of all the blues would be equal to sum of all the whites in each column.

- **Constraint 7:** $blue_{i,j} \leq y_{i,j}$, for $i=1$ to S , $j=1$ to S

This constraint implies that if the cell $[i, j]$ in the *blue* matrix is 1 (i.e. if the cell $[i, j]$ is prefilled with blue) then x value of corresponding $[i, j]$ cell must be 1.

- **Constraint 8:** $white_{i,j} \leq (1 - y_{i,j})$, for $i=1$ to S , $j=1$ to S

This constraint implies that if the cell $[i, j]$ in the *white* matrix is 1 (i.e. if the cell $[i, j]$ is prefilled with white) then x value of corresponding $[i, j]$ cell must be 1. (Robert Fourer)

- **Constraint 9:** $y_{i,j}$ is binary , for $i=1$ to S , $j=1$ to S

Next section converts the above model into an AMPL readable syntax. The AMPL model file will remain the same and can be used to solve any N-In-A-Row puzzle. (Phase I Report) (Moreno-Centeno, Phase I Model)

AMPL Model:

```
#parameters

param S;                #Number of rows/columns in the grid (both must be equal
                        #for a square matrix)
param n;                #Invalid consecutive cells in a row/column
param blue{i in 1..S,j in 1..S};    #Initial blue cells
param white{i in 1..S,j in 1..S};    #Initial white cells

#decision variables

var y{i in 1..S,j in 1..S}, binary;    #binary decision variable whose value is
                                        #1 if blue cell and 0 if white cell

#objective function

maximize colored_cells: 0;

#constraints

subject to

ColInRow_Blue{i in 1..S, j in 1..S-(n-1)}: sum{k in 0..n-1} y[i, j+k] <= n-1;
#no more than n-1 consecutive blue cells in each row

RowInCol_Blue{i in 1..S-(n-1), j in 1..S}: sum{k in 0..n-1} y[i+k, j] <= n-1;
#no more than n-1 consecutive blue cells in each column

ColInRow_White{i in 1..S, j in 1..S-(n-1)}: sum{k in 0..n-1} y[i, j+k] >= 1;
#no more than n-1 consecutive white cells in each row

RowInCol_White{i in 1..S-(n-1), j in 1..S}: sum{k in 0..n-1} y[i+k, j] >= 1;
```

```
#no more than n-1 consecutive white cells in each column

Equal_Row{i in 1..S}: sum{j in 1..S} y[i,j] = S/2;
#each row has an equal number of blue and white cells

Equal_Column{j in 1..S}: sum{i in 1..S} y[i,j] = S/2;
#each column has an equal number of blue and white cells

Prefilled_Blue{i in 1..S,j in 1..S}: blue[i,j] <= y[i,j];
#if blue[i,j] is 1 then x[i,j] is 1

Prefilled_White{i in 1..S,j in 1..S}: white[i,j] <= (1 - y[i,j]);
#if white[i,j] is 1 then (1 - y[i,j]) is 0
```

Problem Statement:

For a better understanding, we illustrate the methodology to solve the 3 in a row puzzle with AMPL, using the following example:

The initial grid, with prefilled blue, white and gray cells is as below:

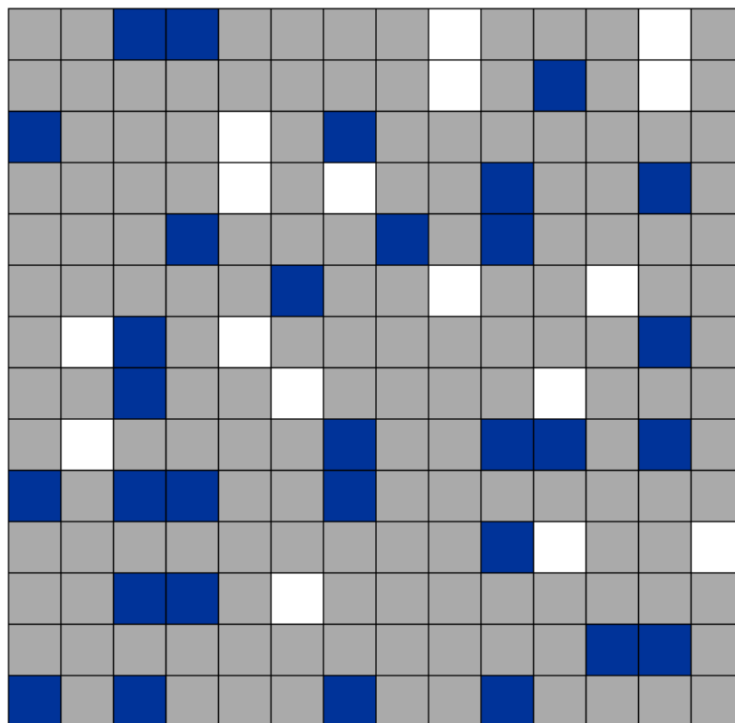


Figure 6-Initial grid of the puzzle

AMPL Data:

```
#parameters

param S:= 14;
#Number of rows/columns in the grid (both have to be equal for a square matrix)

param n:= 3;                                #Invalid consecutive cells in a row/column

#Initial blue cells have an entry 1, rest are 0

param blue: 1 2 3 4 5 6 7 8 9 10 11 12 13 14:=
    1  0 0 1 1 0 0 0 0 0 0 0 0 0 0
    2  0 0 0 0 0 0 0 0 0 0 0 1 0 0
    3  1 0 0 0 0 0 0 1 0 0 0 0 0 0
    4  0 0 0 0 0 0 0 0 0 0 1 0 0 1
    5  0 0 0 1 0 0 0 0 1 0 1 0 0 0
    6  0 0 0 0 0 0 1 0 0 0 0 0 0 0
    7  0 0 1 0 0 0 0 0 0 0 0 0 0 1
    8  0 0 1 0 0 0 0 0 0 0 0 0 0 0
    9  0 0 0 0 0 0 0 1 0 0 1 1 0 1
    10 1 0 1 1 0 0 1 0 0 0 0 0 0 0
    11 0 0 0 0 0 0 0 0 0 0 1 0 0 0
    12 0 0 1 1 0 0 0 0 0 0 0 0 0 0
    13 0 0 0 0 0 0 0 0 0 0 0 0 1 1
    14 1 0 1 0 0 0 1 0 0 1 0 0 0 0;

#Initial white cells have an entry 1, rest are 0

param white: 1 2 3 4 5 6 7 8 9 10 11 12 13 14:=
    1  0 0 0 0 0 0 0 0 0 1 0 0 0 1
    2  0 0 0 0 0 0 0 0 0 1 0 0 0 1
    3  0 0 0 0 1 0 0 0 0 0 0 0 0 0
    4  0 0 0 0 1 0 1 0 0 0 0 0 0 0
    5  0 0 0 0 0 0 0 0 0 0 0 0 0 0
    6  0 0 0 0 0 0 0 0 0 1 0 0 1 0
    7  0 1 0 0 1 0 0 0 0 0 0 0 0 0
    8  0 0 0 0 0 1 0 0 0 0 1 0 0 0
    9  0 1 0 0 0 0 0 0 0 0 0 0 0 0
    10 0 0 0 0 0 0 0 0 0 0 0 0 0 0
    11 0 0 0 0 0 0 0 0 0 0 1 0 0 1
    12 0 0 0 0 0 1 0 0 0 0 0 0 0 0
    13 0 0 0 0 0 0 0 0 0 0 0 0 0 0
    14 0 0 0 0 0 0 0 0 0 0 0 0 0 0;

check: n <= S;                                #Invalid consecutive cells should be less than the grid size

check: S mod 2 == 0;                          #S should be an even number
```


How to insert the data in the AMPL data file?

In order to solve the 3 in a row puzzle, using the AMPL software, the instructions below should be followed:

- The initial grid has 14 columns and 14 rows, so the parameter S is equal to 14. The value of the parameter S changes according to the grid's size.
- The current problem statement requires to solve a 3 in a row problem. Thus, parameter n is equal to 3. This parameter must be changed per the problem's statement. (i.e. 5 in a row puzzle implies $n=5$).
- The initial grid has prefilled blue and white cells, the gray cells are the ones that must be filled respecting the puzzle's rules.
- The parameter *blue* is defined as a 14x14 matrix which must be filled in a way that only the blue cells (in the initial grid) take a value of 1, the white and gray cells take a value of 0. For instance:
 - The cell located in 1st row and 3rd column in the initial grid is blue, so fill up the parameter *blue* matrix as 1 in 1st row and 3rd column and follow suit for other blue cells.
 - The cell located in 3rd row and 5th column in the initial grid is white, so fill up the parameter *blue* matrix as 0 in 3rd row and 5th column and follow suit for other white cells.
 - The cell located in 5th row and 2nd column in the initial grid is gray, so fill up the parameter *blue* matrix as 0 in 5th row and 2nd column and follow suit for other gray cells.
- The parameter *white* is defined as a 14x14 matrix which must be filled in a way that only the white cells (in the initial grid) take a value of 1, the blue and gray cells take a value of 0. For instance:
 - The cell located in 1st row and 3rd column in the initial grid is blue, so fill up the parameter *white* matrix as 0 in 1st row and 3rd column and follow suit for other blue cells.
 - The cell located in 3rd row and 5th column in the initial grid is white, so fill up the parameter *white* matrix as 1 in 3rd row and 5th column and follow suit for other white cells.
 - The cell located in 5th row and 2nd column in the initial grid is gray, so fill up the parameter *white* matrix as 0 in 5th row and 2nd column and follow suit for other gray cells.

N.B: Before uploading the data file in AMPL, two conditions must be checked:

- The value of the parameter n must always be less or equal to the grid's size. (In our example $(n = 3) \leq (S = 14)$)
- The size of the grid S must always be an even number (evenly divisible by 2).

AMPL Output:

```

y  [*,*]
:   1   2   3   4   5   6   7   8   9  10  11  12  13  14 :=
1   0   0   1   1   0   1   0   1   0   1   0   1   0   1
2   0   1   0   0   1   1   0   1   0   0   1   1   0   1
3   1   1   0   1   0   0   1   0   1   0   1   0   1   0
4   1   0   1   0   0   1   0   0   1   1   0   0   1   1
5   0   1   0   1   1   0   1   1   0   1   0   1   0   0
6   1   1   0   0   1   1   0   1   0   0   1   0   0   1
7   0   0   1   1   0   0   1   0   1   1   0   1   1   0
8   0   1   1   0   1   0   0   1   1   0   0   1   0   1
9   1   0   0   1   0   1   1   0   0   1   1   0   1   0
10  1   0   1   1   0   0   1   0   1   0   1   0   0   1
11  0   1   0   0   1   1   0   1   0   1   0   1   1   0
12  1   0   1   1   0   0   1   0   1   0   1   0   0   1
13  0   1   0   0   1   1   0   1   1   0   0   1   1   0
14  1   0   1   0   1   0   1   0   0   1   1   0   1   0;

```

The above output shows the value of the decision variable $y_{i,j}$ in each cell of a 14x14 matrix. Here 1's depict blue color and 0's depict white color. This matrix shows a unique solution to the puzzle and fits all the criteria as well.

If the encoded numbers are decoded into two different colors blue and white, then this is how the solved matrix would look like:

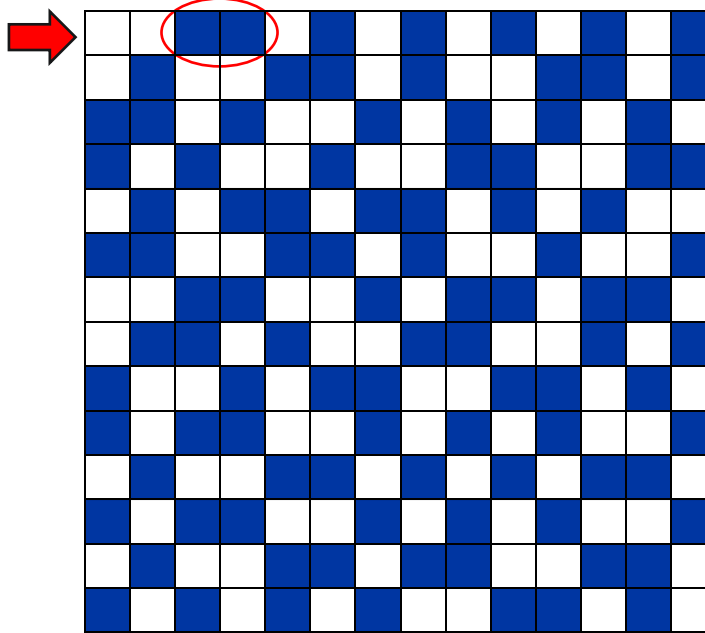


Figure 7-Solution to the puzzle (1)

The 1st row in the above image (highlighted) shows how the final solution would look like. This row satisfies all the requirements of the puzzle, i.e., this row has equal number of blues and whites, there are no more than 2 consecutive blues (as shown by the red circle) or whites in a row

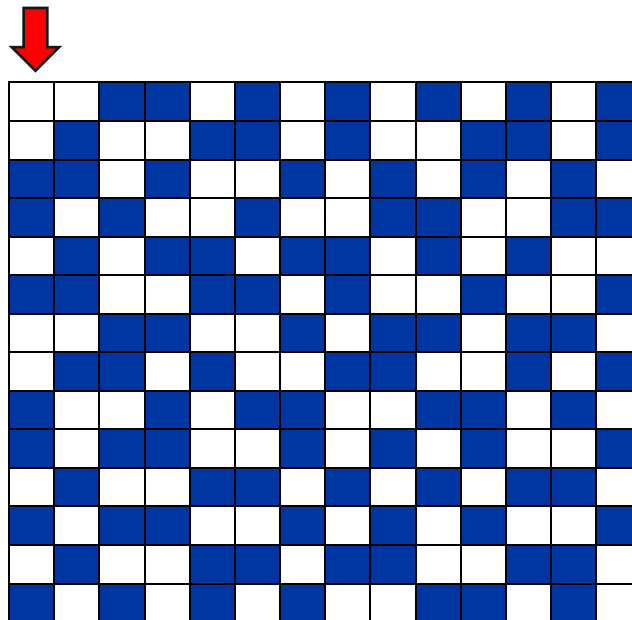


Figure 8-Solution to the puzzle (2)

The 1st column in the above image (highlighted) shows the how the final solution would look like. This column satisfies all the requirements of the puzzle, i.e., this row has equal number of blues and whites, there are no more than 2 consecutive blues or whites in a row

Puzzle 2: ABC Path

Overview:

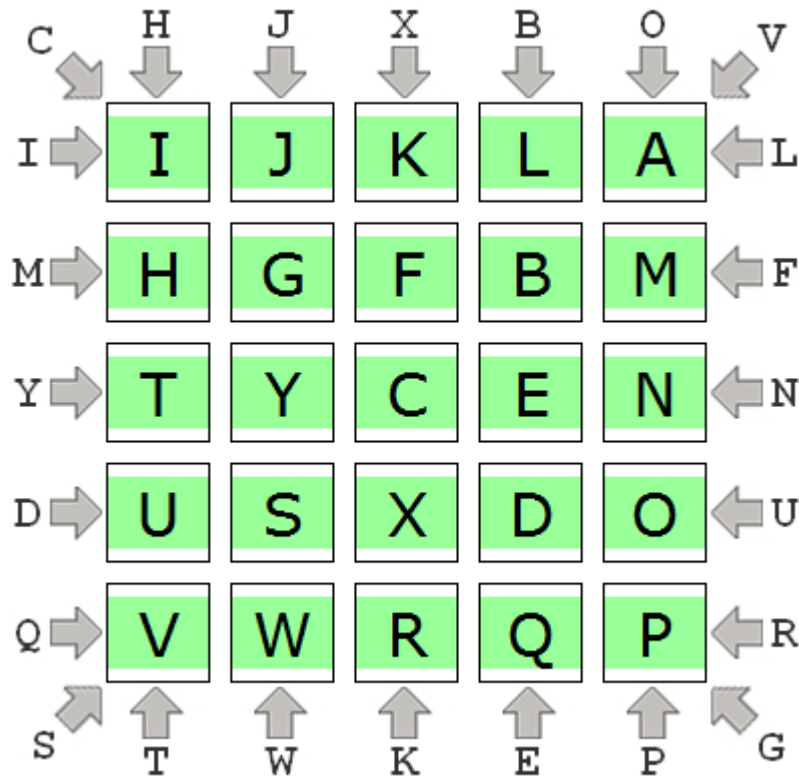


Figure 9- Layout of ABC Path puzzle

ABC Path is a logic puzzle with simple rules that can be summarized as below:

- Enter every alphabet from A to Y into the grid.
- Each alphabet is adjacent to the previous alphabet either horizontally, vertically or diagonally.
- The clues around the edge tell you which row, column or diagonal each alphabet is in.

How to solve ABC Path:

Let's take as an example the ABC path puzzle below, where A is already pre-placed.

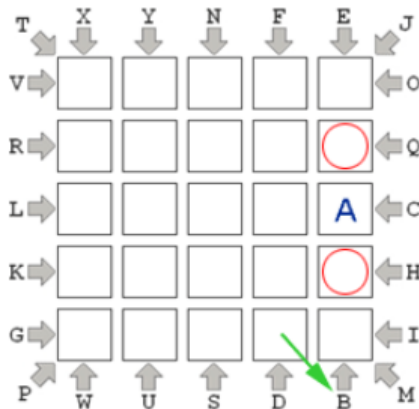


Figure 10 Next step in the puzzle

Step 1 (Figure 10): There are only 2 possible squares for the alphabet B, as it must be in the 5th column.

Step 2 (Figure 11): Wherever the B is, this is the only square where the alphabet C can go.

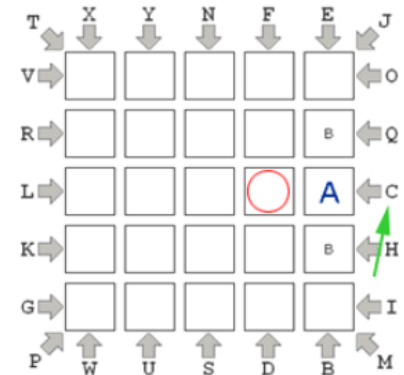


Figure 11 Next step in the puzzle

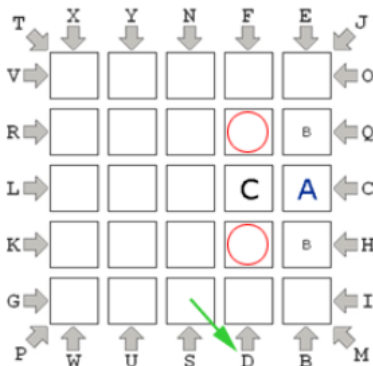


Figure 12 Next step in the puzzle

Step 3 (Figure 12): The alphabet D must now be in one of these two squares.

Final step (Figure 13): This technique can be continued around the rest of the grid, till all the squares are filled following the previously listed rules. (Daily ABC Path, n.d.) (Phase I Report)



Figure 13 Final solved puzzle

IP Model:

Parameters:

dl = A set that contains elements in the left diagonal.

dr = A set that contains elements in the right diagonal.

N = Number of rows/columns.

n = Total number of alphabets to be placed.

ia = Row number where alphabet A has been placed.

ja = Column number where alphabet A has been placed.

t_j = Hints at the top of the grid, for $j=1..N$.

b_j = Hints at the bottom of the grid, for $j=1..N$.

l_i = Hints at the left of the grid, for $i=1..N$.

r_i = Hints at the right of the grid, for $i=1..N$.

Decision Variables:

$$x_{h,i,j} = \begin{cases} 1 & \text{if the } h^{th} \text{ alphabet is in the } i^{th} \text{ row and } j^{th} \text{ column} \\ 0 & \text{otherwise} \end{cases}$$

, for $h=1$ to n , $i=0$ to $N+1$, $j=0$ to $N+1$.

Objective Function:

Maximize ABCPath: 0

Constraints:

The position of letter A:

$$x_{1,ia,ja} = 1$$

Each alphabet is adjacent to the previous one horizontally, vertically or diagonally:

$$x_{h,i,j} + x_{h+1,i,j} \leq \sum_{e=-1}^1 \sum_{f=-1}^1 x_{h+1,i+e,j+f}, \text{ for } h=1 \text{ to } n-1, i=1 \text{ to } N, j=1 \text{ to } N$$

A cell cannot contain more than one letter:

$$\sum_{h=1}^n x_{h,i,j} = 1, \text{ for } i=1 \text{ to } N, j=1 \text{ to } N$$

Each alphabet is placed at a certain cell of the grid:

$$\sum_{i=1}^N \sum_{j=1}^N x_{h,i,j} = 1, \text{ for } h=1 \text{ to } n$$

There are no alphabets placed in the 0th column of the grid:

$$x_{h,i,0} = 0, \text{ for } h=1 \text{ to } n, i=0 \text{ to } N+1$$

There are no alphabets placed in the 0th row of the grid:

$$x_{h,0,j} = 0, \text{ for } h=1 \text{ to } n, j=0 \text{ to } N+1$$

There are no alphabets placed in the N+1th column of the grid:

$$x_{h,i,N+1} = 0, \text{ for } h=1 \text{ to } n, i=0 \text{ to } N+1$$

There are no alphabets placed in the N+1th row of the grid:

$$x_{h,N+1,j} = 0, \text{ for } h=1 \text{ to } n, j=0 \text{ to } N+1$$

Alphabets in the top hint are at the designated columns:

$$\sum_{i=1}^N x_{t,j,i,j} = 1, \text{ for } j=1 \text{ to } N$$

Alphabets in the bottom hint are at the designated columns:

$$\sum_{i=1}^N x_{b,j,i,j}=1 \quad , \text{ for } j=1 \text{ to } N$$

Alphabets in the left hint are at the designated rows:

$$\sum_{j=1}^N x_{l,i,i,j}=1 \quad , \text{ for } i=1 \text{ to } N$$

Alphabets in the right hint are at the designated rows:

$$\sum_{j=1}^N x_{r,i,i,j}=1 \quad , \text{ for } i=1 \text{ to } N$$

Alphabets in the left diagonal are at the designated left diagonal cells:

$$\sum_{i=1}^N x_{h,i,i} = 1 \quad , \text{ for } h \in dl$$

Alphabets in the right diagonal are at the designated right diagonal cells:

$$\sum_{i=1}^N x_{h,i,N+1-i} = 1 \quad , \text{ for } h \in dr$$

Binary requirement:

$$x_{h,i,j} \text{ is binary} \quad , \text{ for } h=1 \text{ to } n, i=0 \text{ to } N+1 \text{ and } j=0 \text{ to } N+1 \text{ (Phase I Report)}$$

(Moreno-Centeno, Phase I Model)

Detailed Description of the Model:

Parameters

Following are the parameters used for solving the puzzle:

- Parameter N (number of rows/columns)
- Parameter n (total number of alphabets to be placed in the cells of the grid)
- Parameter ia (row number of alphabet A)
- Parameter ja (column number of alphabet A)
- Parameter t (alphabet hints on the top of the grid)
- Parameter b (alphabet hints on the bottom of the grid)
- Parameter l (alphabet hints on the left of the grid)
- Parameter r (alphabet hints on the right of the grid)
- Set dl (list of alphabet hints in the left diagonal)
- Set dr (list of alphabets hints in the right diagonal)

All the parameters apart from n , N will change for each new puzzle and would have to be assigned values manually in the specified format in order to be fed into the model file.

Now, let's examine the model file and try to understand the objective function, decision variables, and its constraints.

Decision variables:

$x_{h,i,j}$ is a binary variable, that ensures that a certain alphabet is positioned at a certain cell in the grid.

Objective Function:

This LP is not related to a maximization of profit or a minimization of cost. Thus, the 0 defined in our function simply refers to the fact that the model is to be solved based only on the defined constraints.

Constraints:

- **Constraint 1:** $x_{1,ia,ja} = 1$

This constraint ensures Alphabet A's position in the grid. Here $h=1$ and ia and ja are given parameters (refer above for explanation).

- **Constraint 2:** $x_{h,i,j} + x_{h+1,i,j} \leq \sum_{e=-1}^1 \sum_{f=-1}^1 x_{h+1,i+e,j+f}$,
for $h=1$ to $n-1$, $i=1$ to N , $j=1$ to N

This constraint enumerates over all the values of h , i and j which refers respectively to the alphabets from A to Y, row number and column number. Since we have an $h+1$ in the equation, we have enumerated h from 1 to 24 ($n-1$) instead of 1 to 25 (n). The constraint ensures that each alphabet is adjacent to the previous alphabet either horizontally, vertically or diagonally. The right side of the equation has been summed over $i-1$ to $i+1$ and $j-1$ to $j+1$ over all i and j values ranging from 0 to 5. The term $x_{h+1,i,j}$ is added on the left side of equation to cancel out this term on the right side as the previous alphabet occupies that position. The remaining 8 terms on the right side, indicating all the adjacent possibilities of the consecutive alphabet ($h+1$). The inequality in the constraint ensures that if h^{th} alphabet is in cell i, j then $(h+1)^{th}$ alphabet will be in one of the adjacent cells (RHS is 1 when LHS is 1).

- **Constraint 3:** $\sum_{h=1}^n x_{h,i,j} = 1$, for $i=1$ to N , $j=1$ to N

This constraint enumerates over each cell in the grid (25 cells) and is summed across all the values of h which are alphabets A to Y. It ensures that no two alphabets are in the same cell. So, one unique alphabet per cell.

- **Constraint 4:** $\sum_{i=1}^N \sum_{j=1}^N x_{h,i,j} = 1$, for $h=1$ to n

This constraint enumerates over all the values of h which are alphabets A to Y and is summed across each cell in the grid (25 cells). It ensures that each alphabet is placed in some cell in the grid.

- **Constraint 5:** $x_{h,i,0} = 0$, for $h=1$ to n , $i=0$ to $N+1$

This constraint enumerates over all the values of h which are alphabets A to Y and across row 0 to 6. It ensures that there are no alphabets added in the 0th column of the grid. In the right-hand side of the second constraint, we are subtracting 1 from the column number in some cases, leading to a case where column number is 0. To mitigate this issue, we have defined this constraint.

- **Constraint 6:** $x_{h,0,j} = 0$, for $h=1$ to n , $j=0$ to $N+1$

This constraint is like the fifth constraint, the difference being it is enumerated across columns 0 to 6. This constraint mitigates the case when row number is 0.

- **Constraint 7:** $x_{h,i,N+1} = 0$, for $h=1$ to n , $i=0$ to $N+1$

This constraint enumerates over all the values of h which are alphabets A to Y and across row 0 to 6. It ensures that there are no alphabets added in the 6th column of the grid. In the right-hand side of the second constraint, we are adding 1 to the column number in some cases, leading to a case where the column number is 6. To mitigate this issue, we have defined this constraint.

- **Constraint 8:** $x_{h,N+1,j} = 0$, for $h=1$ to n , $j=0$ to $N+1$

This constraint is like the seventh constraint, the difference being it is enumerated across columns 0 to 6. This constraint mitigates the case when row number is 6.

- **Constraint 9:** $\sum_{i=1}^N x_{t_j,i,j} = 1$, for $j=1$ to N

This constraint enumerates over all the columns in the grid and is summed across all the rows in the grid. The first subscript t_j in decision variable x pulls the j^{th} entry in the t row matrix. The contents of t are the alphabets that have hints across the columns of the grid. The constraint ensures that each alphabet in t row matrix has its column value fixed as per its position in the matrix (hint) and the row value be anything between 1 to 5. The right-hand side is equal to one ensuring that each alphabet in t row matrix is only in one cell.

- **Constraint 10:** $\sum_{i=1}^N x_{b_j,i,j} = 1$, for $j=1$ to N

This constraint is like the previous constraint. The only difference is the first subscript here is b_j in decision variable x . So, this constraint takes care of the location of alphabets listed in the b row matrix.

- **Constraint 11:** $\sum_{j=1}^N x_{l_i,i,j}=1$, for $i=1$ to N

This constraint enumerates over all the rows in the grid and is summed across all the columns in the grid. The first subscript l_i in decision variable x pulls the i^{th} entry in the l row matrix. The contents of l are the alphabets that have hints across the rows of the grid. The constraint ensures that each alphabet in l row matrix has its row value fixed as per its position in the matrix (hint) and the column value be anything between 1 to 5. The right-hand side is equal to one ensuring that each alphabet in l row matrix is only in one cell.

- **Constraint 12:** $\sum_{j=1}^N x_{r_i,i,j}=1$, for $i=1$ to N

This constraint is like the 11th constraint. The only difference is the first subscript here is r_i in decision variable x . So, this constraint takes care of the location of the alphabets listed in the r row matrix.

- **Constraint 13:** $\sum_{i=1}^N x_{h,i,i} = 1$, for $h \in dl$

This constraint enumerates over the set dl which contains the alphabets associated with the left diagonal (hint). Each alphabet in dl must satisfy the above condition that is the row number and column number of the cell are equal ensuring it's in the left diagonal of the grid. The right-hand side is equal to one ensuring that each alphabet in dl is only in one cell.

- **Constraint 14:** $\sum_{i=1}^N x_{h,i,N+1-i} = 1$, for $h \in dr$

This constraint enumerates over the set dr which contains the alphabets associated with the right diagonal (hint). Each alphabet in dr must satisfy the above condition that is sum of row number and column number of the cell is equal to 6 ensuring it's in the right diagonal of the grid. The right-hand side is equal to one ensuring that each alphabet in dr is only in one cell. (Robert Fourer)

- **Constraint 15:** $x_{h,i,j}$ is binary , for $h=1$ to n , $i=0$ to $N+1$ and $j=0$ to $N+1$
(Phase I Report) (Moreno-Centeno, Phase I Model)

Below section converts the above mentioned model into AMPL readable syntax. This model has been generalized to accommodate different grid size problems as well.

AMPL Model:

```
#model

#parameters

set dl;                #elements in the left diagonal
set dr;                #elements in the right diagonal
param N;               #number of rows/columns
param n;               #total number of alphabets to be placed
param ia;              #row number of alphabet A
param ja;              #column number of alphabet A
param t{j in 1..N};   #hints at the top
param b{j in 1..N};   #hints at the bottom
param l{i in 1..N};   #hints at the left
param r{i in 1..N};   #hints at the right

#decision variable

var x{h in 1..n, i in 0..N+1, j in 0..N+1} binary;    #binary decision variable, h
indicates the alphabets, i row number and j column number

#obj function

maximize Abcpath: 0;

#constraints

subject to

Position_A: x[l, ia, ja] = 1;    #alphabet A position given in the problem

Location_Constraint{h in 1..n-1, i in 1..N, j in 1..N}:
x[h, i, j]+x[h+1,i,j]<= sum{e in -1..1, f in -1..1} x[h+1,i+e,j+f];

#location condition, each alphabet can be in the 8 surrounding cells of previous
alphabet cell ensuring consecutive alphabet being next to each other
One_Alphabet{i in 1..N, j in 1..N}: sum{h in 1..n} x[h, i, j] = 1;
#exactly one alphabet on each cell

All_fill{h in 1..n}: sum{i in 1..N, j in 1..N} x[h, i, j] = 1;
#each alphabet is placed in some cell of the grid

Zeroth_pos_column{h in 1..n, i in 0..N+1}: x[h, i, 0] = 0;
#no alphabet is in column number 0

Zeroth_pos_row{h in 1..n, j in 0..N+1}: x[h, 0, j] = 0;
#no alphabet is in row number 0

Sixth_pos_column{h in 1..n, i in 0..N+1}: x[h, i, N+1] = 0;
#no alphabet is in column number N+1

Sixth_pos_row{h in 1..n, j in 0..N+1}: x[h, N+1, j] = 0;
#no alphabet is in row number N+1

Top{j in 1..N}: sum{i in 1..N} x[t[j], i, j] = 1;
#alphabets in the top hint are at the designated column
```

```

Bottom{j in 1..N}: sum{i in 1..N} x[b[j], i, j] = 1;
#alphabets in the bottom hint are at the designated column

Left{i in 1..N}: sum{j in 1..N} x[l[i], i, j] = 1;
#alphabets in the left hint are at the designated row

Right{i in 1..N}: sum{j in 1..N} x[r[i], i, j] = 1;
#alphabets in the right hint are at the designated row

Diagonal_Left{h in dl}: sum{i in 1..N} x[h, i, i] = 1;
#alphabets in the left diagonal are at the designated diagonal

Diagonal_Right{h in dr}: sum{i in 1..N} x[h, i, N+1-i] = 1;
#alphabets in the right diagonal are at the designated diagonal

```

Problem Statement:

For a better understanding, we illustrate the methodology to solve the ABC Path puzzle with AMPL, using the following example:

The initial grid, with hints for each row, column and diagonal, is as below:

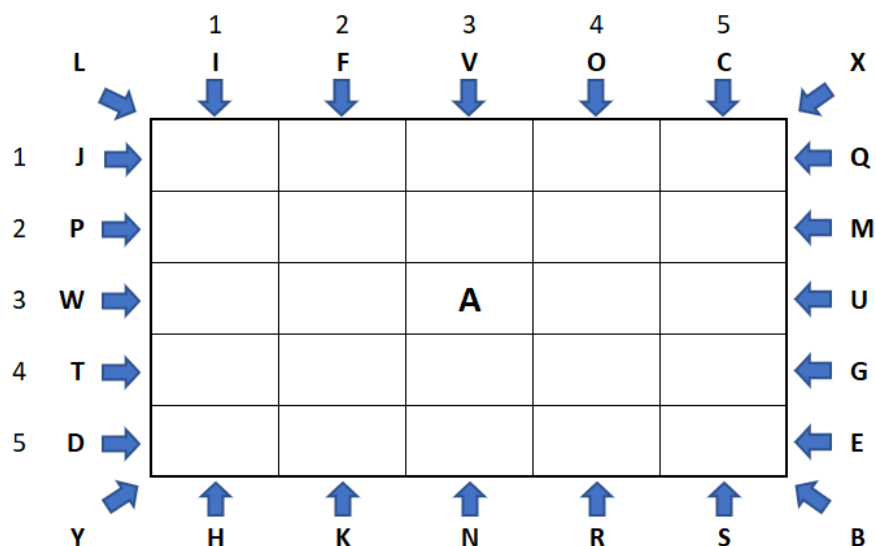


Figure 14-Initial problem ABC Path

AMPL Data:

```
#data

#parameters and sets

param N:= 5;      #number of rows/columns
param n:= 25;     #total number of alphabets to be placed
param ia:= 3;     #row number of alphabet A
param ja:= 3;     #column number of alphabet A

param t:= 1 9      #hints at the top
          2 6
          3 22
          4 15
          5 3;

param b:= 1 8      #hints at the bottom
          2 11
          3 14
          4 18
          5 19;

param l:= 1 10     #hints at the left
          2 16
          3 23
          4 20
          5 4;

param r:= 1 17     #hints at the right
          2 13
          3 21
          4 7
          5 5;

set dl:= 12, 2;    #alphabets in the left diagonal
set dr:= 24, 25;   #alphabets in the right diagonal

check: n=N^2;      #ensuring the relation between n and N
                  #parameter is maintained
```

How to insert the data in the AMPL data file?

In order to solve the ABC Path puzzle, using the AMPL software, the instructions below should be followed:

- The grid has 5 columns and 5 rows, so the parameter N is equal to 5. The value of the parameter N is changed according to the grid's size.
- The current problem statement requires to fill 25 alphabets; thus, parameter n is equal to N^2 that is 25.
- The initial grid has alphabet A position in the grid at 3rd row and 3rd column which are ia and ja parameters respectively.
- The hints provided in the puzzle are defined by 4 parameters (top, bottom, left, right hints) and 2 sets (left, right diagonal hints). The alphabets have been encoded as numbers ranging from 1 to 25 for A to Y.
 - Parameter t is a column matrix of size $N \times 1$. Each row corresponds to the encoded number for the corresponding alphabet in the top hint. For example, top hint contains I, F, V, O, C (in order) and the corresponding encoded numbers are 9, 6, 22, 15, 3. The encoded numbers are filled in the column matrix in such a way that the row number corresponds to the position of the top hint. For example, encoded number of alphabet I is filled in row 1 of the column matrix t as the alphabet I hint is the first column of the main grid.
 - Like parameter t , parameter b is filled with encoded numbers of the alphabets in the order as show in Figure 15-21.
 - Like parameter t , parameter l is filled with encoded numbers of the alphabets in the order as shown in Figure 15-21. The only difference is that the encoded numbers are filled in such a way the row number in the column matrix l corresponds to the row number hint of the main grid. For example, alphabet P is filled in row 2 of the column matrix l as the alphabet P hint is the second row of the main grid.
 - Like parameter l , parameter r is filled with encoded numbers of the alphabets in the order as show in Figure15-21.
 - Set dl includes the list of encoded numbers of the alphabets in the left diagonal hint as shown in Figure 15-21. In the current puzzle, the alphabets in the left diagonal hint are L and B.
 - Set dr includes the list of encoded numbers of the alphabets in the right diagonal hint as shown in Figure 15-21. In the current puzzle, the alphabets in the right diagonal hint are X and Y.

AMPL Output:

The matrices printed below is a copy of the output of the model. Basically, it's the print of the decision variable obtained using AMPL.

```
x ['0,']
:      0      1      2      3      4      5      6 :=
1      0      0      0      0      0      0      0
2      0      0      0      0      0      0      0
3      0      0      0      0      0      0      0
4      0      0      0      0      0      0      0
5      0      0      0      0      0      0      0
6      0      0      0      0      0      0      0
7      0      0      0      0      0      0      0
8      0      0      0      0      0      0      0
9      0      0      0      0      0      0      0
10     0      0      0      0      0      0      0
11     0      0      0      0      0      0      0
12     0      0      0      0      0      0      0
13     0      0      0      0      0      0      0
14     0      0      0      0      0      0      0
15     0      0      0      0      0      0      0
16     0      0      0      0      0      0      0
17     0      0      0      0      0      0      0
18     0      0      0      0      0      0      0
19     0      0      0      0      0      0      0
20     0      0      0      0      0      0      0
21     0      0      0      0      0      0      0
22     0      0      0      0      0      0      0
23     0      0      0      0      0      0      0
24     0      0      0      0      0      0      0
25     0      0      0      0      0      0      0
```

Figure 15-Puzzle output 0th Row

```
['1,']
:      0      1      2      3      4      5      6 :=
1      0      0      0      0      0      0      0
2      0      0      0      0      0      0      0
3      0      0      0      0      0      0      0
4      0      0      0      0      0      0      0
5      0      0      0      0      0      0      0
6      0      0      0      0      0      0      0
7      0      0      0      0      0      0      0
8      0      0      0      0      0      0      0
9      0      0      0      0      0      0      0
10     0      1      0      0      0      0      0
11     0      0      1      0      0      0      0
12     0      0      0      0      0      0      0
13     0      0      0      0      0      0      0
14     0      0      0      1      0      0      0
15     0      0      0      0      1      0      0
16     0      0      0      0      0      0      0
17     0      0      0      0      0      1      0
18     0      0      0      0      0      0      0
19     0      0      0      0      0      0      0
20     0      0      0      0      0      0      0
21     0      0      0      0      0      0      0
22     0      0      0      0      0      0      0
23     0      0      0      0      0      0      0
24     0      0      0      0      0      0      0
25     0      0      0      0      0      0      0
```

Figure 16-Puzzle output 1st Row

```
['2,']
:      0      1      2      3      4      5      6 :=
1      0      0      0      0      0      0      0
2      0      0      0      0      0      0      0
3      0      0      0      0      0      0      0
4      0      0      0      0      0      0      0
5      0      0      0      0      0      0      0
6      0      0      0      0      0      0      0
7      0      0      0      0      0      0      0
8      0      0      0      0      0      0      0
9      0      1      0      0      0      0      0
10     0      0      0      0      0      0      0
11     0      0      0      0      0      0      0
12     0      0      1      0      0      0      0
13     0      0      0      1      0      0      0
14     0      0      0      0      0      0      0
15     0      0      0      0      0      0      0
16     0      0      0      0      0      1      0
17     0      0      0      0      0      0      0
18     0      0      0      0      1      0      0
19     0      0      0      0      0      0      0
20     0      0      0      0      0      0      0
21     0      0      0      0      0      0      0
22     0      0      0      0      0      0      0
23     0      0      0      0      0      0      0
24     0      0      0      0      0      0      0
25     0      0      0      0      0      0      0
```

Figure 17-Puzzle output 2nd Row

```
['3,']
:      0      1      2      3      4      5      6 :=
1      0      0      0      1      0      0      0
2      0      0      0      0      0      0      0
3      0      0      0      0      0      0      0
4      0      0      0      0      0      0      0
5      0      0      0      0      0      0      0
6      0      0      0      0      0      0      0
7      0      0      0      0      0      0      0
8      0      1      0      0      0      0      0
9      0      0      0      0      0      0      0
10     0      0      0      0      0      0      0
11     0      0      0      0      0      0      0
12     0      0      0      0      0      0      0
13     0      0      0      0      0      0      0
14     0      0      0      0      0      0      0
15     0      0      0      0      0      0      0
16     0      0      0      0      0      0      0
17     0      0      0      0      0      0      0
18     0      0      0      0      0      0      0
19     0      0      0      0      0      1      0
20     0      0      0      0      0      0      0
21     0      0      0      0      1      0      0
22     0      0      0      0      0      0      0
23     0      0      1      0      0      0      0
24     0      0      0      0      0      0      0
25     0      0      0      0      0      0      0
```

Figure 18-Puzzle output 3rd Row

```

['.4,']
:      0      1      2      3      4      5      6 :=
1      0      0      0      0      0      0      0
2      0      0      0      0      1      0      0
3      0      0      0      0      0      0      0
4      0      0      0      0      0      0      0
5      0      0      0      0      0      0      0
6      0      0      0      0      0      0      0
7      0      1      0      0      0      0      0
8      0      0      0      0      0      0      0
9      0      0      0      0      0      0      0
10     0      0      0      0      0      0      0
11     0      0      0      0      0      0      0
12     0      0      0      0      0      0      0
13     0      0      0      0      0      0      0
14     0      0      0      0      0      0      0
15     0      0      0      0      0      0      0
16     0      0      0      0      0      0      0
17     0      0      0      0      0      0      0
18     0      0      0      0      0      0      0
19     0      0      0      0      0      0      0
20     0      0      0      0      0      1      0
21     0      0      0      0      0      0      0
22     0      0      0      1      0      0      0
23     0      0      0      0      0      0      0
24     0      0      1      0      0      0      0
25     0      0      0      0      0      0      0

```

Figure 19-Puzzle output 4th Row

```

['.5,']
:      0      1      2      3      4      5      6 :=
1      0      0      0      0      0      0      0
2      0      0      0      0      0      0      0
3      0      0      0      0      0      1      0
4      0      0      0      0      1      0      0
5      0      0      0      1      0      0      0
6      0      0      1      0      0      0      0
7      0      0      0      0      0      0      0
8      0      0      0      0      0      0      0
9      0      0      0      0      0      0      0
10     0      0      0      0      0      0      0
11     0      0      0      0      0      0      0
12     0      0      0      0      0      0      0
13     0      0      0      0      0      0      0
14     0      0      0      0      0      0      0
15     0      0      0      0      0      0      0
16     0      0      0      0      0      0      0
17     0      0      0      0      0      0      0
18     0      0      0      0      0      0      0
19     0      0      0      0      0      0      0
20     0      0      0      0      0      0      0
21     0      0      0      0      0      0      0
22     0      0      0      0      0      0      0
23     0      0      0      0      0      0      0
24     0      0      0      0      0      0      0
25     0      1      0      0      0      0      0

```

Figure 20-Puzzle output 5th Row

```

['.6,']
:      0      1      2      3      4      5      6 :=
1      0      0      0      0      0      0      0
2      0      0      0      0      0      0      0
3      0      0      0      0      0      0      0
4      0      0      0      0      0      0      0
5      0      0      0      0      0      0      0
6      0      0      0      0      0      0      0
7      0      0      0      0      0      0      0
8      0      0      0      0      0      0      0
9      0      0      0      0      0      0      0
10     0      0      0      0      0      0      0
11     0      0      0      0      0      0      0
12     0      0      0      0      0      0      0
13     0      0      0      0      0      0      0
14     0      0      0      0      0      0      0
15     0      0      0      0      0      0      0
16     0      0      0      0      0      0      0
17     0      0      0      0      0      0      0
18     0      0      0      0      0      0      0
19     0      0      0      0      0      0      0
20     0      0      0      0      0      0      0
21     0      0      0      0      0      0      0
22     0      0      0      0      0      0      0
23     0      0      0      0      0      0      0
24     0      0      0      0      0      0      0
25     0      0      0      0      0      0      0
;

```

Figure 21-Puzzle output 6th Row

The decision variable x has 3 indices; alphabet, row number, and column number as explained above. Since, it's a binary variable, the matrices above are filled with 1 or 0. There are 7 matrices, one for every row ranging from 0 to 6. The rows of the matrix are the alphabets and the columns are the column numbers of the main grid ranging from 0 to 6. Since 0th and 6th row is empty as per our constraints, the 1st and the 7th matrix are filled with 0's in the cells. Barring, the 0 and 7th matrix, the remaining matrices that is, 2nd to 6th matrix each have five 1's and rest 0's in the cell. The column 0 and 6 in each matrix is filled with 0's as there are no alphabets in column 0 and 6 of the main grid. This implies that there are exactly 5 alphabets in each row (which is our desired output). The 1's indicate the position of the alphabet (corresponding row number of the matrix), row number in the main grid (matrix number) and column number in the main grid (corresponding column number in the matrix). For example, the green and red circles are the row number and column number for alphabet F (number 6 – blue circle) respectively in the main grid (in Figure 22).

	['5']						
:	0	1	2	3	4	5	6 :=
1	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0
3	0	0	0	0	0	1	0
4	0	0	0	0	1	0	0
5	0	0	0	1	0	0	0
6	0	0	1	0	0	0	0
7	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0
25	0	1	0	0	0	0	0

Figure 22-Puzzle explanation example 5th Row

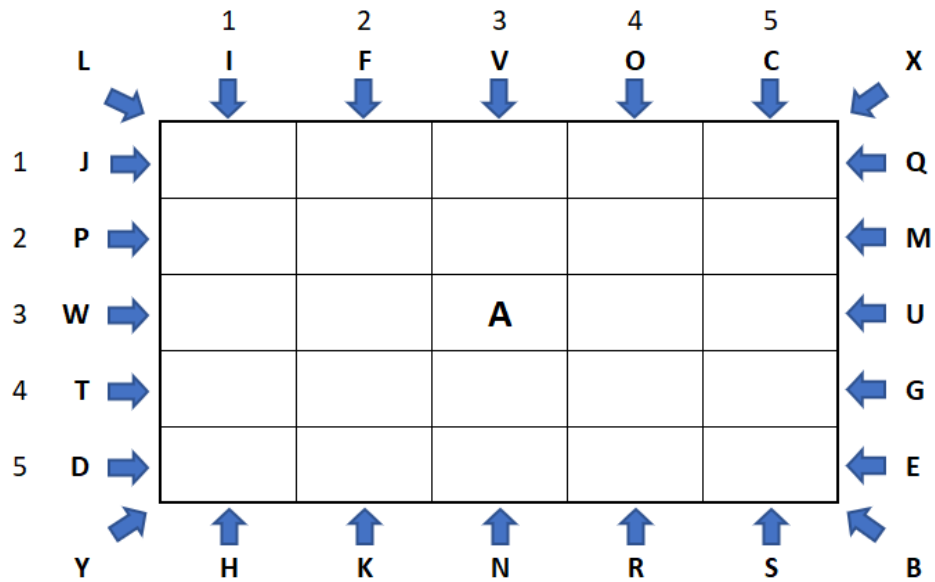


Figure 23-Puzzle to be solved

Figure 23 is a grid representation of the puzzle to be solved, with the position of A specified (3,3).

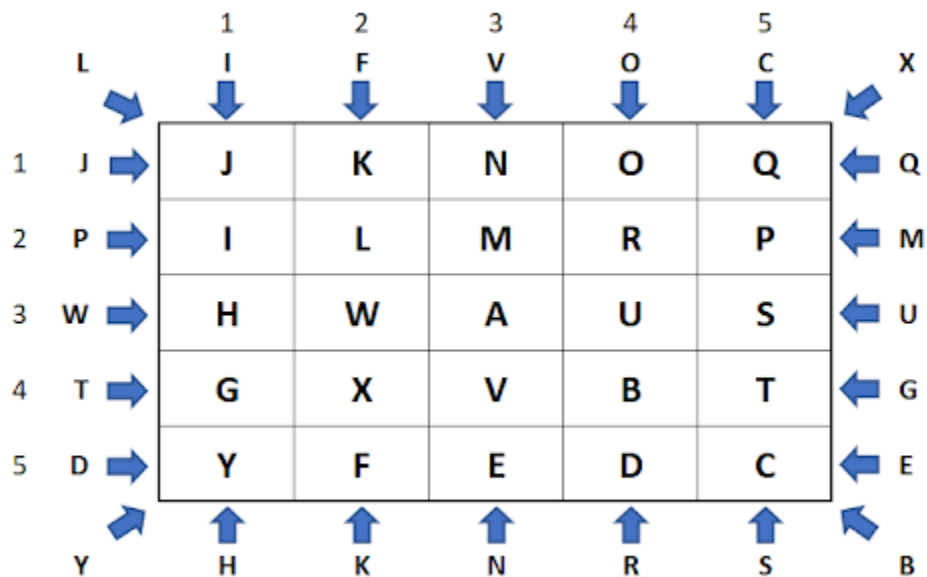


Figure 24-Solution to the puzzle

Figure 24 is the solution to the puzzle. The display matrices as shown in Figure 15-21 are used to fill the cells in the grid.

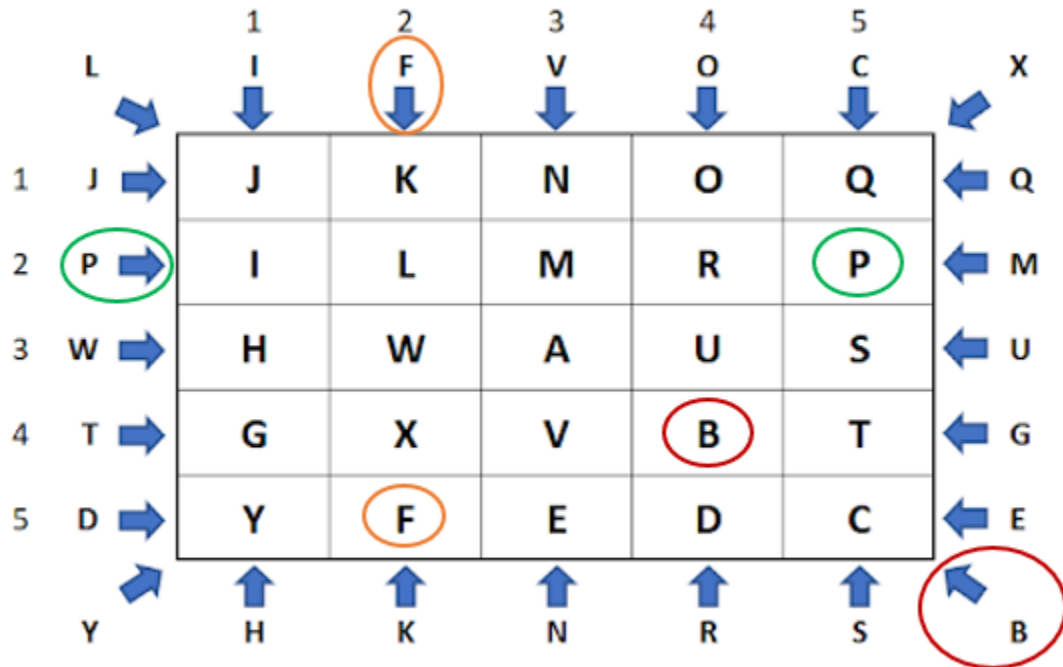


Figure 25-Explanation of Puzzle Solution

As it can be seen in Figure 25:

- Alphabet B is in 4th row and 4th column of the grid. The hint and the position are highlighted with red circles. The hint suggests that alphabet B must be in the left diagonal of the grid. The position of B satisfies the condition.
- Alphabet F is in 5th row and 2nd column of the grid. The hint and the position are highlighted with orange circles. The hint suggests that alphabet F must be in the 2nd column of the grid. The position of F satisfies the condition.
- Alphabet P is in 2nd row and 5th column of the grid. The hint and the position are highlighted with green circles. The hint suggests that alphabet P must be in the 2nd row of the grid. The position of P satisfies the condition.

Similarly, the rest of the alphabets are filled in the grid, each satisfying the hint provided in the puzzle.

Technical Explanation

The report till now talked about how to write mathematical equations, and how to input those equations in AMPL to get solutions to the puzzles. But the solutions in AMPL are a bit difficult to interpret, so we leverage AMPL API (Application program interface) using Python which allows a program to be manipulated using some basic form of coding. (AMPL API Documentation, n.d.)

Using this API, one can fetch all the data, its parameters, sets, variables, and objective function. One can leverage advanced coding concepts to get user input and manipulate it in order to feed into the AMPL data file.

In our case, we ask the user to input all the parameters in a simplified form (letters instead of the numbers in the grid) and make the puzzle-solving user friendly. These letters are then converted to numbers using Python code, which is AMPL readable format.

We can solve the model within the Python code itself using all the functions that are generally available in AMPL. `Ampl.solve()` produces results in numerical format, which again might be in a multi-dimensional array (as in the case of ABC Path) format. So, we take help of Python to convert those multi-dimensional arrays into easy to interpret format (letters in case of ABC Path) and create a 2-dimensional grid showing the exact location of letters (in case of ABC Path) or label colors (in case of 3 in a row).

We have written a generic code which can solve either of the puzzles and can change the parameters as well. For ease of use, we have created a batch file which on a double click launches the python code file and thus can be used by users of all age, even with just a basic understanding of computers.

The Python code can be found in the appendix attached at the end of this document.

User Manual

Please double click on the shortcut named “Puzzle solver” in your desktop, in order to launch the solver.



in your

Once the solver is operational, the following window will pop up, allowing to choose the puzzle you want to solve.

```
Enter the kind of puzzle you want to solve :
Enter 1 for 3-In-A-Row, or 2 for ABC-Path:
```

Figure 26-Choose puzzle to solve

3-In-A-Row Puzzle:

- **STEP 1:** Define the type of puzzle to be solved. If you are solving a 3-In-A-Row Puzzle, please type ‘1’ and hit enter on your keyboard.

```
Enter 1 for 3-In-A-Row, or 2 for ABC-Path: 1
```

Figure 27-Solve 3 in a Row

- **STEP 2:** Define the size of the grid. Since the initial grid contains 14 rows and 14 columns, please type ‘14’ and hit enter right after. If the grid contained 10 rows and 10 columns, you would have to enter ‘10’.

```
Enter the size of the grid (that is number of rows or columns in the grid) :
```

```
14
```

Figure 28-Enter size of the grid

- **STEP 3:** Enter the number of minimum invalid consecutive cells. Since you are trying to solve a 3-In-A-Row Puzzle, you must type ‘3’ and hit enter. For instance: If you are trying to solve a 5-In-A-Row Puzzle, please type ‘5’. In a general rule, if you are solving an **N**-In-A-Row Puzzle, please type **N** and hit enter.

```
Enter the number of minimum invalid consecutive cells :
```

```
3
```

Figure 29-Enter number of invalid consecutive cells

- **STEP 4:** Fill the cells with their predefined colors as per the initial grid. For that, please respect carefully the following instructions:
 - You must enter the colors in each row starting from the first row till the last one.
 - You must enter hints based on the cells in each row, starting from the left to the right.
 - Please input the letters in the following way: G for a gray cell, B for a blue cell and W for a white one, in capital or small letters.

For example, the cells in the first row of the initial grid are prefilled as below:



Figure 30-Hints in the first row in 3-in-a-row puzzle

Starting from the left, the 2 first cells are gray; thus, you have to write GG, the third and fourth cells are Blue, which correspond to B and so on.

```
Enter hints when asked for, in the following format
Enter B for Blue, W for White and G for Grey
Input the hints for row - 1 :
GGBBGGGGWGGGWG
```

Figure 31-Input hints when prompted

Please continue entering the prefilled cells following the methodology described above.

```

Input the hints for row - 2 :
GGGGGGGGWGBGWG
Input the hints for row - 3 :
BGGGWGBGGGGGGG
Input the hints for row - 4 :
GGGGWGWGGBGGBG
Input the hints for row - 5 :
GGGBGGGBGBGGGG
Input the hints for row - 6 :
GGGGGBGGWGGWGG
Input the hints for row - 7 :
gWdgwgggggggpg
Input the hints for row - 8 :
GGBGGWGGGGWGGG
Input the hints for row - 9 :
GWGGGGBGGBBBBG
Input the hints for row - 10 :
BGBBGGBGGGGGGG
Input the hints for row - 11 :
GGGGGGGGGBWGGW
Input the hints for row - 12 :
GGBBGWGGGGGGGG
Input the hints for row - 13 :
GGGGGGGGGGGBBG
Input the hints for row - 14 :
BGBGGGBGGBGGGG

```

Figure 32-Hints for the whole puzzle

Once you have filled all the hints for every row, you will automatically get the answer to the puzzle you're seeking to solve.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	White	White	Blue	Blue	White	Blue	White	Blue	White	Blue	White	Blue	White	Blue
2	Blue	Blue	White	White	Blue	Blue	White	Blue	White	White	Blue	Blue	White	White
3	Blue	White	White	Blue	White	White	Blue	White	Blue	White	Blue	White	Blue	Blue
4	White	Blue	Blue	White	White	Blue	White	White	Blue	Blue	White	White	Blue	Blue
5	White	Blue	White	Blue	Blue	White	Blue	Blue	White	Blue	White	Blue	White	White
6	Blue	White	Blue	White	Blue	Blue	White	Blue	White	White	Blue	White	Blue	White
7	White	Blue	White	Blue	White	White	Blue	White	Blue	Blue	White	Blue	White	Blue
8	White	Blue	Blue	White	Blue	White	White	Blue	Blue	White	White	Blue	White	Blue
9	Blue	White	White	Blue	White	Blue	Blue	White	White	Blue	Blue	White	Blue	White
10	Blue	White	Blue	Blue	White	White	Blue	White	Blue	White	Blue	White	White	Blue
11	White	Blue	White	White	Blue	Blue	White	Blue	White	Blue	White	Blue	Blue	White
12	Blue	White	Blue	Blue	White	White	Blue	White	Blue	White	Blue	White	White	Blue
13	White	Blue	White	White	Blue	Blue	White	Blue	Blue	White	White	Blue	Blue	White
14	Blue	White	Blue	White	Blue	White	Blue	White	White	Blue	Blue	White	Blue	White

Figure 33-Solved 3-in-a-row puzzle

The output obtained through the solver, can be translated into the following final filled grid, in which: **White** in the solver refers to a white a cell, and **Blue** simply means that the cell must be filled in blue.

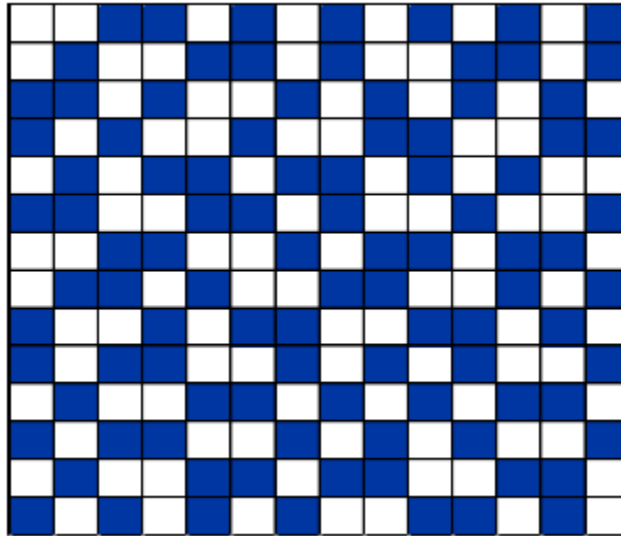
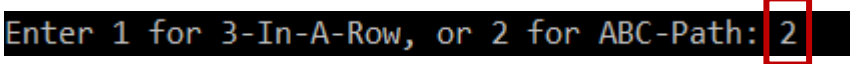


Figure 34-Actual puzzle after replacing words with corresponding colors

ABC Path Puzzle:

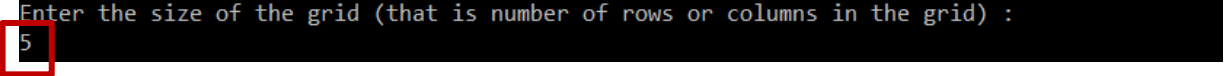
- **STEP 1:** For solving an ABC Path Puzzle, please type '2' and hit enter on your keyboard.



```
Enter 1 for 3-In-A-Row, or 2 for ABC-Path: 2
```

Figure 35-Input 2 for solving ABC Path

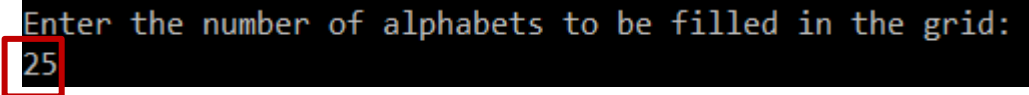
- **STEP 2:** Define the size of the grid. As the size of the grid for the ABC Path puzzle never changes, please **always** type '5' and hit enter right after.



```
Enter the size of the grid (that is number of rows or columns in the grid) : 5
```

Figure 36-Enter size of grid

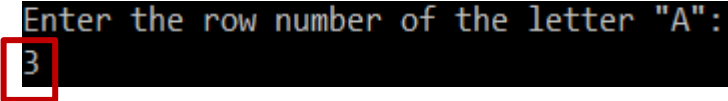
- **STEP 3:** Enter the number of alphabets to be filled in the grid, since the letters that are to be included are from A to Y, the number to enter in this step must **always** be '25' and hit enter.



```
Enter the number of alphabets to be filled in the grid: 25
```

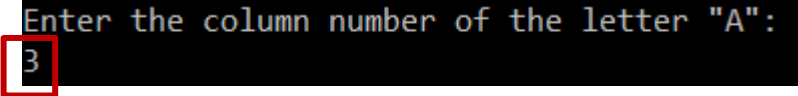
Figure 37-Enter number of alphabets to be filled

- **STEP 4:** Enter the row number, as well as the column number of letter A, that is always preplaced in the grid. In the example below, A is in the center of the grid, placed in the 3rd row and 3rd column, so you must type 3 in the row number and 3 in the column number, and hit enter.



```
Enter the row number of the letter "A": 3
```

Figure 38-Input the row number where 'A' is located



```
Enter the column number of the letter "A": 3
```

Figure 39-Input the column number where 'A' is located

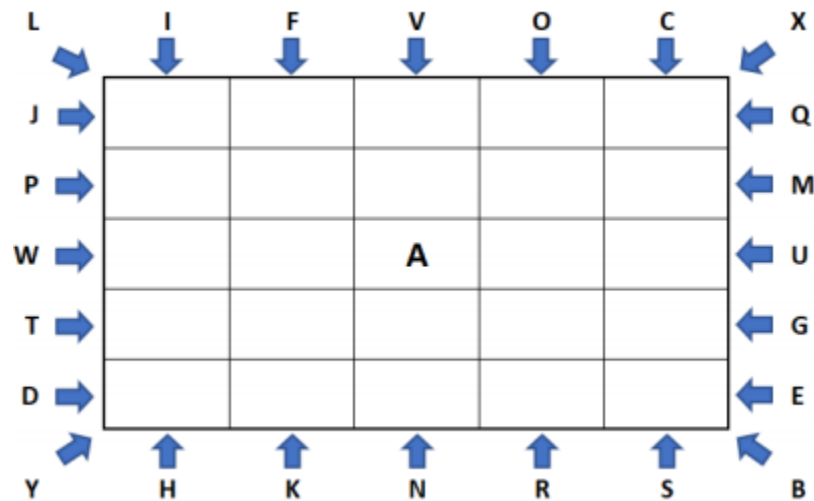


Figure 40-Layout of the ABC Path puzzle

- **STEP 5:** Enter the hints which refer to the letters surrounding the grid, from the left to right in the top and bottom of the grid, and from the top to the bottom in the right and left of the grid, excluding the 4 points in the 4 corners of the grid, and hit enter. For instance, in the top of the grid, we have the following hints: IFVOC. The hints can be entered in capital or small letters.

```

Enter the hints (letters shown at top) at the top of the grid:
IFVOC
Enter the hints (letters shown at bottom) at the bottom of the grid:
HKNRS
Enter the hints (letters shown at left) at the left side of the grid:
JPWTD
Enter the hints (letters shown at right) at the right side of the grid:
QMUGE

```

Figure 41-Hints for the ABC Path puzzle

- **STEP 6:** Complete the insertion of the hints, by entering the 4 points in the 4 corners of the grid. Starting from the corner letter in the top left of the grid, followed by the corner letter in the bottom right of the grid, and hit enter. In this example, those letters would be: LB.

After that, you should enter the corner letter in the top right of the grid, followed by the corner letter in the bottom left of the grid, and hit enter. Which are in this case: XY.

```

Enter the letters shown at the top left and bottom right side of the diagonal in the grid:
LB
Enter the letters shown at the top right and bottom left side of the diagonal in the grid:
XY

```

Figure 42-Hints on the diagonals of the grid

Once, you have entered all the hints and hit enter, the solution to the ABC Path puzzle will automatically be displayed.

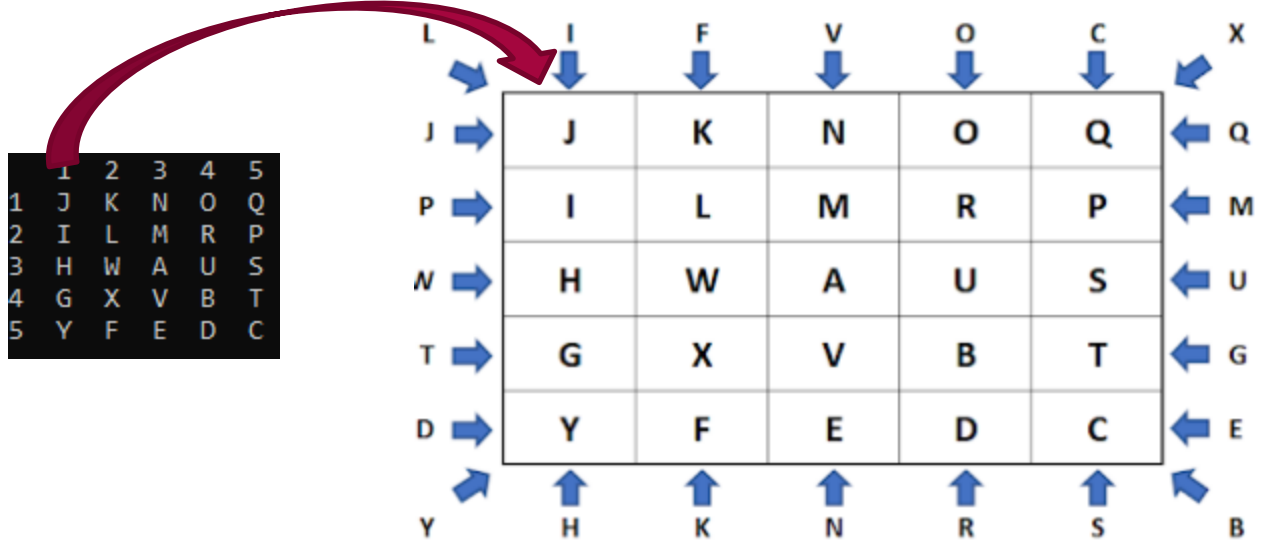


Figure 43-Interpretation of the solution of the puzzle

All you have to do now is to fill your grid based on the solution displayed by the solver, while respecting the order of the alphabets in the grid. For example: the letter J in the first row/first column must be placed in the top left cell in the grid and so on.

Conclusion

The puzzles, 3-In-A-Row and ABC path are solved using a Binary Integer Linear Programming Model. This report uses a scalable IP model, ensuring that the model files are flexible and can be used for any data, represented in the format as discussed earlier and meeting the puzzle conditions. The model is fed to AMPL along with the data containing the parameter information, resulting in desired results as output. Python code helps generalize the model and makes it easy to input and change the data file. Python code also helps print the solution in an easy-to-interpret format.

References

- (n.d.). Retrieved from Wikipedia: <https://en.wikipedia.org/wiki/AMPL>
- (n.d.). Retrieved from AMPL API Documentation: <https://ampl.com/api/nightly/python/index.html>
- Daily 3-In-A-Row*. (n.d.). Retrieved from BrainBashers:
<https://www.brainbashers.com/show3inarow.asp?date=1030&size=6&diff=2>
- Daily ABC Path*. (n.d.). Retrieved from BrainBashers: <http://www.brainbashers.com/showabcpath.asp>
- Introductory Guide on Linear Programming*. (n.d.). Retrieved from Analytics Vidhya:
<https://www.google.com/url?q=https://www.analyticsvidhya.com/blog/2017/02/introductory-guide-on-linear-programming-explained-in-simple-english/&sa=D&source=hangouts&ust=1573733977519000&usg=AFQjCNHG9mjT9sg9-HKOi-83RTrC8YeRLA>
- Kira Goldner, R. K. (n.d.). *Nonokenken: Solving Puzzles Using Integer Programs*. Retrieved from
<https://pdfs.semanticscholar.org/6ad4/bb8301adbef70dde52772941658d9e40ebbe.pdf>
- Moreno-Centeno, E. (n.d.). *Instructions on AMPL and CPLEX*.
- Moreno-Centeno, E. (n.d.). *Phase I Model*.
- (n.d.). *Phase I Report*.
- Robert Fourer, D. M. (n.d.). *AMPL: A Modeling Language for Mathematical Programming*. Cengage Learning.

Appendix

```

import amplpy
import pandas as pd
from string import ascii_lowercase
from amplpy import AMPL, Environment

print("Enter the kind of puzzle you want to solve : \n")

puzzle = int(input("Enter 1 for 3-In-A-Row, or 2 for ABC-Path: "))

ampl = AMPL()

ampl.setOption("solver", "cplex")

if puzzle == 1:

    ampl.reset()
    ampl.read('p2t05mod3.mod')
    ampl.readData('p2t05dat3.dat')

    #ask for user input for 3 in a row
    size = int(input("Enter the size of the grid (that is number of rows or
columns in the grid) : \n"))

    invalid = int(input("Enter the number of minimum invalid consecutive cells :
\n"))

    lst = []
    lst_lst = []

    print("Enter hints when asked for, in the following format \n")
    print("Enter B for Blue, W for White and G for Grey \n")

    for each in range(size):
        lst_lst = list(input(f"Input the hints for row - {each+1} : \n",))
        lst.append(lst_lst)

    S = ampl.getParameter('S')
    S.getValues()
    S.setValues([size])

    n = ampl.getParameter('n')
    n.getValues()
    n.setValues([invalid])

    index0_lst = [i+1 for i in range(size)]
    index1_lst = [i+1 for i in range(size)]

    #Fetch and change the blue parameter

    lst_blue = []
    for each in range(len(lst)):
        lst_blue.append([1 if x == 'B' else 0 for x in lst[each]])

    blue = ampl.getParameter('blue')

    blue.getValues()

```



```

blue.setValues({
    (index0, index1): lst_blue[i][j]
    for i, index0 in enumerate(index0_lst)
    for j, index1 in enumerate(index1_lst)
})

#Fetch and change the white parameter

lst_white = []
for each in range(len(lst)):
    lst_white.append([1 if x == 'W' else 0 for x in lst[each]])

white = ampl.getParameter('white')

white.getValues()

white.setValues({
    (index0, index1): lst_white[i][j]
    for i, index0 in enumerate(index0_lst)
    for j, index1 in enumerate(index1_lst)
})

try:
    invalid <= size
except:
    raise ValueError('Number of invalid consecutive cells should be less than
the size of the grid')

try:
    size%2 == 0
except:
    raise ValueError('Size of the grid should be a multiple of 2')

ampl.solve()

y = ampl.getVariable('y')
y_df = y.getValues().toPandas()

y = ampl.getVariable('y')
y_df = y.getValues().toPandas()
y_df=y_df.astype(int)
y_df.index=pd.MultiIndex.from_tuples(y_df.index,names=('row','column'))
y_df['row']=y_df.index.get_level_values(0).astype(int)
y_df['column']=y_df.index.get_level_values(1).astype(int)
y_output=y_df.pivot(index='row',columns='column',values='y.val')
y_output.replace([1,0],['Blue','White'],inplace=True)
y_output.columns.name = None
y_output.index.name = None
print(y_output)

else:

    ampl.reset()
    ampl.read('p2t05modA.mod')
    ampl.readData('p2t05datA.dat')

    #ask for user input for abc path

```

```

    size = int(input("Enter the size of the grid (that is number of rows or
columns in the grid) : \n"))

    num_letters = int(input("Enter the number of alphabets to be filled in the
grid: \n"))

    row_A = int(input("Enter the row number of the letter \"A\": \n"))
    col_A = int(input("Enter the column number of the letter \"A\": \n"))

    top = list(input("Enter the hints (letters shown at top) at the top of the
grid: \n").lower())
    bottom = list(input("Enter the hints (letters shown at bottom) at the bottom
of the grid: \n").lower())
    left = list(input("Enter the hints (letters shown at left) at the left side of
the grid: \n").lower())
    right = list(input("Enter the hints (letters shown at right) at the right side
of the grid: \n").lower())

    left_diag = list(input("Enter the letters shown at the top left and bottom
right side of the diagonal in the grid: \n").lower())
    right_diag = list(input("Enter the letters shown at the top right and bottom
left side of the diagonal in the grid: \n").lower())

    #define the dictionary of alphabets with values as their respective position
    letters = {letter: index for index, letter in enumerate(ascii_lowercase,
start=1)}

    #Fetch and change the parameters

    N = ampl.getParameter('N')
    N.getValues()
    N.setValues([size])

    n = ampl.getParameter('n')
    n.getValues()
    n.setValues([num_letters])

    ia = ampl.getParameter('ia')
    ia.getValues()
    ia.setValues([row_A])

    ja = ampl.getParameter('ja')
    ja.getValues()
    ja.setValues([col_A])

    t = ampl.getParameter('t')
    t.getValues()
    t.setValues([letters[character] for character in top if character in letters])

    b = ampl.getParameter('b')
    b.getValues()
    b.setValues([letters[character] for character in bottom if character in
letters])

    l = ampl.getParameter('l')
    l.getValues()
    l.setValues([letters[character] for character in left if character in
letters])

    r = ampl.getParameter('r')

```

```

r.getValues()
r.setValues([letters[character] for character in right if character in
letters])

dl = ampl.getSet('dl')
dl.getValues()
dl.setValues([letters[character] for character in left_diag if character in
letters])

dr = ampl.getSet('dr')
dr.getValues()
dr.setValues([letters[character] for character in right_diag if character in
letters])

ampl.solve()

x = ampl.getVariable('x')
x_df = x.getValues().toPandas()

try:
    num_letters = size**2
except:
    raise ValueError("Please make sure that number of letters is the square of
the size of the grid")

col_list = []
row_list = []
letter_num_list = []
for letter in range(1, num_letters+1):
    for row in range(1, size+1):
        for col in range(1, size+1):
            if int(x.get(letter, row, col).value()) == 1:
                col_list.append(col)
                row_list.append(row)
                letter_num_list.append(letter)

letter_list = []
for each in letter_num_list:
    letter_list.append(list(letters.keys())[list(letters.values()).index(each)].upper())

index = list(range(1, size+1))
columns = list(range(1, size+1))

df = pd.DataFrame(index = index, columns = columns)
df = df.fillna(0)

for index, value in enumerate(letter_list):
    df.iloc[row_list[index]-1, col_list[index]-1] = value

print(df)

```