# **Assignment 1 (10%)**

**Binary Puzzle Solver with Constraint Satisfaction Algorithms**

DUE DATE: Friday, October 13th, 2023, 11:59 PM ET

**Objective:**

* Develop a Python program that solves a binary puzzle using multiple constraint satisfaction algorithms. Then, evaluate the performance of each algorithm based on different puzzle sizes.

**Assignment Guidelines:**

* All students in this course must read and meet the expectations described in the [Student Academic Integrity](https://intranet.laurentian.ca/policies/2017.09.19%20-%20Policy%20and%20Procedures%20on%20Academic%20Integrity%20-%20EN.pdf).
* Assignments must follow the programming standards document published on the course website in the D2L. Marks will be taken off if standards are not followed.
* **Submit just one .py or .ipynb file AND the associate report in PDF per group**. Name the file based on your group “ID” and the assignment number, exactly as in this example for **assignment 1 and** **group 1**: **COSC\_4117EL\_A1\_G1.py**. Same naming convention applies to the PDF, **COSC\_4117EL\_A1\_G1.pdf**.
* **Do NOT zip the files** that you submit.
* You may submit the assignment multiple times, but only the most recent version will be marked.
* After the due date and time, a late penalty of 2% per hour, or a portion of an hour, will be applied. After 49 hours, the penalty is 100% and submissions will no longer be accepted. The date and time of the last file submitted control the mark for the entire assignment.
* These assignments are your chance to learn the material for the exams. **Code your assignments independently (or solely within your group)**. We use software to compare all submitted assignments with one another, and pursue academic dishonesty vigorously. **You must complete the Honesty Declaration in the D2L before you will be able to submit your assignment.**

**Introduction:**

A binary puzzle is a square (n x n) puzzle with cells that can contain a 0, a 1, or are blank. The objective is to fill in the blanks according to these rules:

1. *Each cell must contain a number either 0 or 1.*
2. *No three consecutive ones or zeros in a row or a column.*
3. *The number of ones and zeros in each row and each column is the same.*
4. *No two rows or two columns can be identical.*

Try the binary puzzles from [binarypuzzle.com](http://www.binarypuzzle.com/index.php) or [puzzle-binairo.com](https://www.puzzle-binairo.com/).

**Note: some binary puzzles may have more than one solution.**

**Part 1: Reading the Puzzle and Print the Solution (5 Marks)**

1. **Reading the Puzzle**

In part 1, you need to implement a function that reads in a binary puzzle from a file. The puzzle will be represented as a matrix where:

* ‘0’s represent zeros.
* ‘1’s represent ones.
* ‘\_‘s represent blank spaces.

A [sample](https://drive.google.com/file/d/1w0qF-TJDndbPpOaSrD0I-SnXbDINwR8o/view?usp=sharing) 6x6 input puzzle is shown below:

| 1\_\_0\_\_  \_\_00\_1  \_00\_\_1  \_\_\_\_\_\_  00\_1\_\_  \_1\_\_00 |
| --- |

1. **Output the Solution**

The output of your program should be at least an *r* \* *c* ASCII output that shows the completed solution (You may use a GUI too). For example, the output of the program for the above puzzle is given as:

| Solution:  101010  010011  100101  011010  001101  110100 |
| --- |

**You may use either list, numpy array, or both.**

**Part 2: Solving the Puzzle as a CSP**

1. **Constraint Satisfaction (20 Marks)**

You need to Implement the puzzle solver using the following constraint satisfaction algorithms:

1. Backtracking
2. Arc Consistency (AC-3)

The easiest way to convert a binary puzzle into a **CSP** is by creating **one variable for each cell**. The **domain of each variable will be either "1" or "0"**. The **constraints for these variables will enforce the rules for the rows and columns of the puzzle**. *However, this representation might not be the most efficient. You should consider exploring other representations and describe them in your report.*

1. **Heuristics (9 Marks)**

Next, you need to incorporate the following three heuristics into the **AC-3 algorithm**. These heuristics pertain to variable selection and can guide the order in which variables (in this case, puzzle cells) are chosen for assignment:

1. **Random Variable (H1)**: This heuristic **randomly** selects an empty cell. (You may already have this from Part 2A)
2. **Most Constrained Variable (H2)**: This heuristic chooses the variable with the fewest legal values remaining (aka. minimum remaining values, MRV). In the context of the binary puzzle, it refers to a cell in a row or column that, due to current assignments, has limited legal options. **Use H1 is tie.**
3. **Most Constraining Variable (H3)**: This heuristic opts for the variable that places the most constraints on its neighboring cells. In the context of a binary puzzle, this can be interpreted as selecting a cell in a row or column which, upon assignment, would most limit the legal assignments of neighboring cells. **Use H1 is tie.**

It's crucial to note that the efficacy of heuristics can vary based on the specific problem instance. In certain scenarios, they can significantly boost performance, while in others, their impact might be negligible or even detrimental.

Your program's output should provide **one** **possible solution to the puzzle (if there is one)**. Display this solution adhering to the output format presented in Part 1.

Additionally, the program should print the total number of nodes and the runtime required to generate the solutions using your algorithms.

# **Part 3: Report (1-2 pages including graphics, single-spacing, 12-pt, style: Times New Roman): (6 Marks)**

You can download the report template from [here](https://docs.google.com/document/d/1cUljySUYkgWuxwJRjMPg3rOeVOh4q3BF7M4Uep_BVc0/edit?usp=sharing).

You will observe how different **combinations of variable and value selection heuristics** (**H1, H2, H3**) can impact the solving speed of the binary puzzle. Discuss the number of **search nodes and runtime** of these combinations for problems of different sizes (**6x6, 8x8, 10x10, and 12x12**). You can find more puzzles at the suggested websites.

**Based on your observation, which algorithm performed better? Why?**

**Submission Guidelines:**

1. Submit your code as a **.py (preferable) or** a **Jupyter Notebook .ipynb** file.
2. You may submit more than one .py file for different algorithms. Make sure you rename the filename accordingly.
3. Submit your report as a **PDF** file, ensure that all plots, figures, and tables are properly labeled.
4. Your assignment should be self-contained, meaning a person should be able to understand your process and results just by reading your report and going through your code.

**Evaluation Criteria:**

1. Correctness of implementation.
2. Efficiency of the algorithms.
3. Quality and clarity of the code and report.
4. Depth of analysis in comparing the algorithms.

**Tips:**

1. Make sure to handle edge cases, such as **when there is no possible solution**.
2. You might want to start by implementing the backtracking algorithm (as required) first as a baseline, then extend your program to include the other methods.

**Note on Group Contributions and Grading:**

If any group member believes that another member of their group deserves a lower grade due to their contribution level, they are encouraged to address this concern. To formalize this, the group can include an additional section in their report detailing the situation and the proposed grade adjustment, with the consent of all group members. It's essential that all group members agree and provide their consent for any proposed grade changes.

It's always best to communicate openly within your group and seek collaborative solutions. However, if discrepancies in contributions are significant and consensus is achieved, this mechanism ensures fairness in grading.

**Best of luck! Remember, the process and learning are as important as the final results.**