Lab 3: Constraint Satisfaction Problem(CSP)

Mirza Mohammad Azwad (Student ID: 200042121)

CSE 4618: Artificial Intelligence
BScEngg in Software Engineering(SWE), Department Computer Science and Engineering, Islamic University of Technology(IUT)

25th March, 2024

I Contents

II Introduction

This lab deals with a special branch of search problems called **Constraint Satisfaction Problems(CSP)**. Unlike the search problems seen in the previous labs, this category of search problem does not try to find an optimal path to a goal but rather the existence of a possible solution that satisfies all the given criteria without much emphasis on the path to get to the solution. Some of the pre-requisites for this lab can be considered the idea of a constraint graph which models all the constraints of the problem in a way such that traversing the graph using certain strategies would yield the result. Next we have the idea of:

- **Unary Constraint**: Constraints involving a single variable
- **Binary Constraint**: Constraint involving 2 variables
- **Higher Order Constraints**: Constraints involving multiple variables

In this lab we were asked to essentially create constraint graphs that could be used to find potential solutions. How the solution is found was kept abstract and we did not have to implement it, However, we did create the graphs that led to the solution and the applet we were provided with could carry out arc consistency and also find the solution while allowing the user to have control over the steps like, stop, backtrack, fine step, step back, etc.

Unlike previous labs we were provided a jar file, and we could run this jar file using a simple *java -jar constraint.java* assuming the jar is saved in the current working directory.

III Analysis of Problems

1 Task I

In the first question, we have four people Zahid(Z), Ishrak(I), Nafisa(N) and Farabi(F) who visit a restaurant and have their own preferences about what to order or not from the items: Paratha(P), Kashmiri Naan(K), Biryani(B) and Special Rice(S). The variables are Z, I, N and F with the domain consisting of P, K, B and S From the given information, we can extract the following constraints:

A Unary Constraints

- $Z \neq P$: Z doesn't like P
- $N \neq K$: N won't order K
- $F = \{B, S\}$: F likes rice items(B and S).

B Binary Constraints

- $I \neq F$: I and F would end up sharing, so wants different dishes
- $Z \neq F$ **and** $Z \neq N$: Z wants a unique dish
- $Z = I$: Z likes to copy I

Upon using these constraints we could solve to find a solution that satisfies all these constraints

2 Task II

In this problem, we have four people looking for an apartment to rent: Ali(A), Sristy(S), Maliha(M) and Rafid(R). The variables are A, S, M and R while domain consists of 1, 2, 3 and 4. There are four floors labeled 1 to 4. Based on their requirements:

A Binary Constraints

- $A \neq S$:A and S cannot live in the same floor
- $(A, M)\epsilon\{(2,2),(1,3),(2,3),(3,1),(3,2)\}$: If A and M live on same floor its 2, otherwise one of them must live on the 3rd floor
- $R \neq S$, $R \neq M$ **and** $R \neq A$: R must not live in the same floor as anyone else
- $R > M$: R must live on a higher floor than M

3 Task III

Here we are tasked with finding a suitable arrangement for 6 people standing in a queue: Rifat(R), Atiq(A), Ishmam(I), Farhan(F), Tabassum(T) and Sabrina(S). The variables are R, A, I, F, T and S with the domain consisting of 1, 2, 3, 4, 5 and 6. There are 6 positions which we can label 1 to 6 and accordingly we can arrange these people while paying heed to the following requirements:

A Unary Constraints

- $T\epsilon\{1, 6\}$: T can either be in the front or the back
- $S = 5$: S has only one person behind her

B Binary Constraints

- $|F - I| = 1$ **and** $|F - A| = 1$: F is between I and A
- $|S - R| = 1$: S and R stands next to each other

C Higher-Order Constraints

- $allDif(R, A, I, F, T, S)$: Noone can stand on the same position in the line

4 Task IV(Home Task)

Here we have two faculties, X and Y, along with 5 courses C,L,Q,D and G. We also have 4 time slots, let's call them 1,2,3 and 4, representing 8am-9am, 9am-10am, 10am-11am, 11am-12am. So what we essentially end up doing is having X1, X2, X3, X4, Y1, Y2, Y3 and Y4 as the domain. The constraints for this problem are relatively complicated. In this problem, it took me a while to figure out how to use a new domain as a cartesian product of the domain {X,Y} representing faculties and the domain {1,2,3,4} to obtain the aforementioned domain.

A Unary Constraints

- **Gϵ X, where X ϵ {X1,X2,X3,X4}**: This implies that only X does G as X is good at this
- **L<=3, where L ϵX1,X2,X3,Y1,Y2,Y3**: L must be started at or before 10am which is slot 3
- **Q<=3, where Q ϵX1,X2,X3,Y1,Y2,Y3**: Q must be started at or before 10am which is slot 3
- **D<3, where Q ϵX1,X2,Y1,Y2**: D must be finished before 10am which is slot 2

B Binary Constraints

- **G<D**: Time when D greater than G
- **C<L**: Time when L greater than C

C Higher-Order Constraints

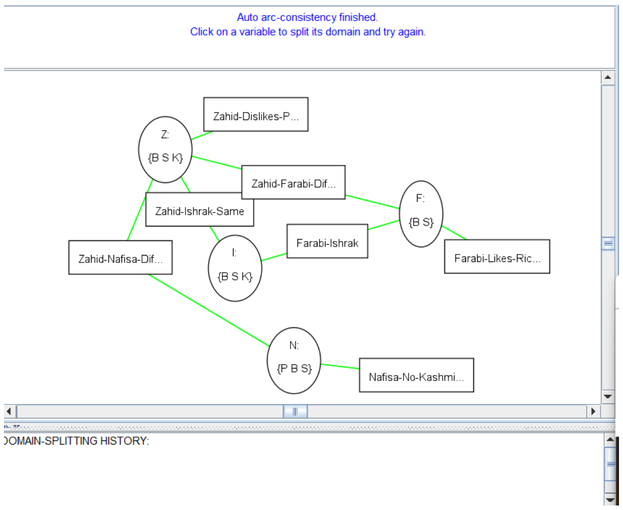
- No Multitasking: No faculty can do two things at the same time
- Q Takes 2: Q takes 2 periods, so no faculty can do anything for the next 2 periods
- L takes 2: Q takes 2 periods, so no faculty can do anything for the next 2 periods
- D involves 2: D involves both the faculties so no faculty can do anything else at that time
- Same Faculty must Take L and C, as AI Lab and Class has to be conducted by the same faculty

Now, another constraint that I ignored is that if X takes DBMS lab, s/he doesn't take the AI Lab, in my idea, it doesn't matter as even if X takes DBMS lab in a scheduling perspective this doesn't matter as both X and Y would be busy while the DBMS lab is going on.

IV Explanation of Solutions

1 Task I

In Task I, most of the constraints were pretty easy and majority of the discussion is covered in the analysis section. I represented easy of these constraints in a descriptive manner such as Zahid-Dislikes-Paratha, which means that Zahid wouldn't order paratha so we select all the cells having Zahid and paratha


Auto arc-consistency finished.
Click on a variable to split its domain and try again.

DOMAIN-SPLITTING HISTORY:

Figure 1. Arc Consistency for Task I

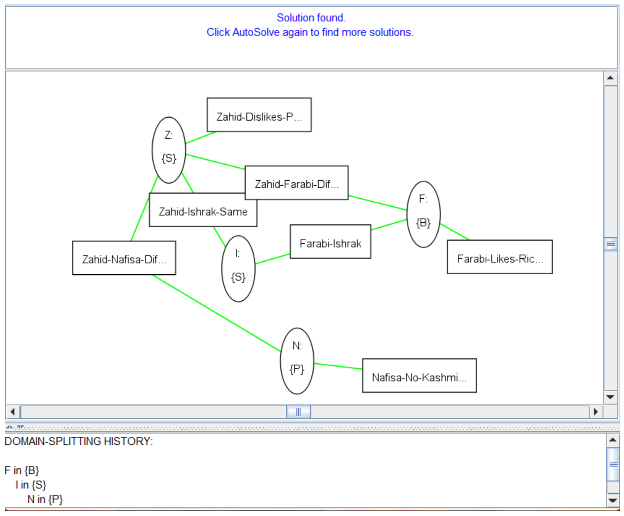
to be false, demonstrating a unary constraint. Similarly Zahid-Ishrak-Same means Z and I would order possibly the same thing so its essentially like Z=I, equality being set, but initially when I started doing this task, I didn't know about the equality, so I set a **Custom** constraint here as well ensuring Z and I has True for the same values within the domain. Similarly Nafisa-No-Kashmiri-Naan means, since she had it, she doesn't want to have it again so its essentially like N≠K. Similarly Farabi-Likes-Rice means that this is a unary constraint, ensuring that Farabi has rice in his domain. Zahid wants a unique dish so Zahid-Nafisa-Different and Zahid-Farabi-Different constraints are given but Zahid also likes to copy Ishrak so Zahid-Ishrak-Same as mentioned before was given.

2 Task II

Task II is also a pretty decent task, where the constraints are simple and self-explanatory, most of them were discussed in the analysis section but the important thing to discuss, is that I combined two of the constraints, Ali and Maliha if they are on the same floor its 2 and if they are on different floors one of them are 3. Typically they can be modified as two seperate constraints, but instead I joined them and selecting the appropriate values as True or False for this pair. Similarly Rafid-Different from everyone is being represented with Rafid-Different. Additionally we have Ali-Sristy-Different meaning Ali and Sristy has to be on different floors and Rafid-High-Maliha implies that Rafid lives on a higher floor than Maliha.

3 Task III

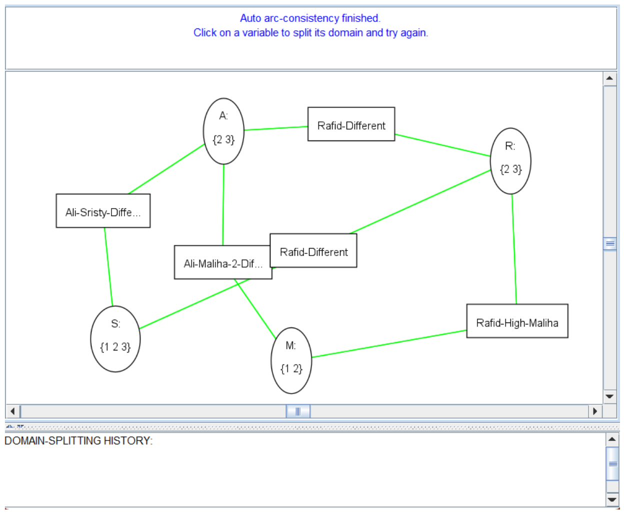
In Task III, I faced some struggle due to the absence of the allDIff constraint, so I made a slight modification, here for each variable I checked to ensure that they are not equal using the **Equality** constraints by taking the **complement**. I did check similar to a mesh topology with a constraint between each of the variables, along with a central constraint that checks takes the complement of R=I=S=F=A=T, this constraint is similar to **not(R==I and I==S and S==F and F==A and A==T)** in pseudocode, and taking the not


Solution found.
Click AutoSolve again to find more solutions.

DOMAIN-SPLITTING HISTORY:

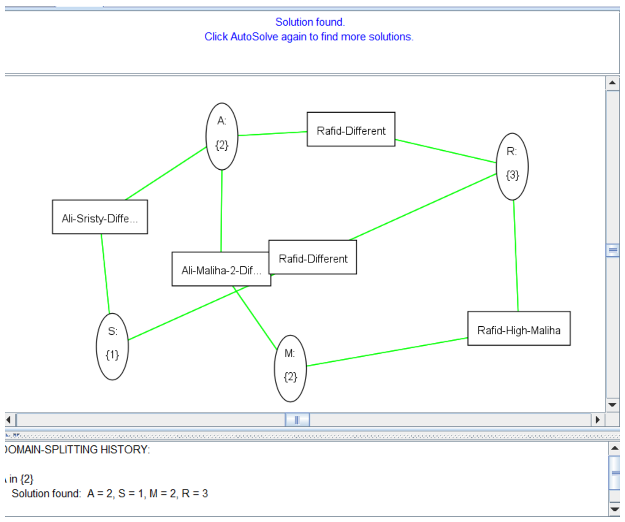
F in {B}
I in {S}
N in {P}

Figure 2. Solution for Task I


Auto arc-consistency finished.
Click on a variable to split its domain and try again.

DOMAIN-SPLITTING HISTORY:

Figure 3. Arc Consistency for Task II


Solution found.
Click AutoSolve again to find more solutions.

DOMAIN-SPLITTING HISTORY:

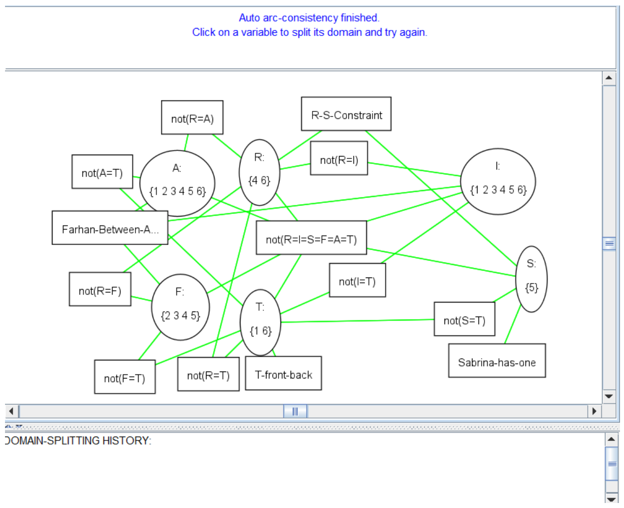
in {2}
Solution found: A = 2, S = 1, M = 2, R = 3

Figure 4. Solution for Task II

of this implies **R!=I or I!=S or S!=F or F!=A or A!=T** as it becomes subject to the De Morgan's Laws. This **or** operation is not satisfactory to check for all the variables, and hence the mesh topology like checking is required. Additionally to represent Farhan-Between-Atiq-and-Ishmam, we essentially select the approrpirate True-False cells such that $|F - I| = 1$ and $|F - A| = 1$ is met. Similarly the T is in front or back implies the values 1 and 6 using the Custom Constraint to be set to True. The constraints for Sabrina involves a unary constraint where only 5 is selected as she is bound to have only one person behind her. Since Rifat and Sabrina must be next to each other we need another Custom Constraint with the appropriate True False values selected representing Rifat and Sabrina as people next to one another via the R-S constraint. In the R-S constraint, only consecutive value pairs are selected as valid or True. The rest of the constraints involve the aforementioned mesh topology-like not equal constraints.

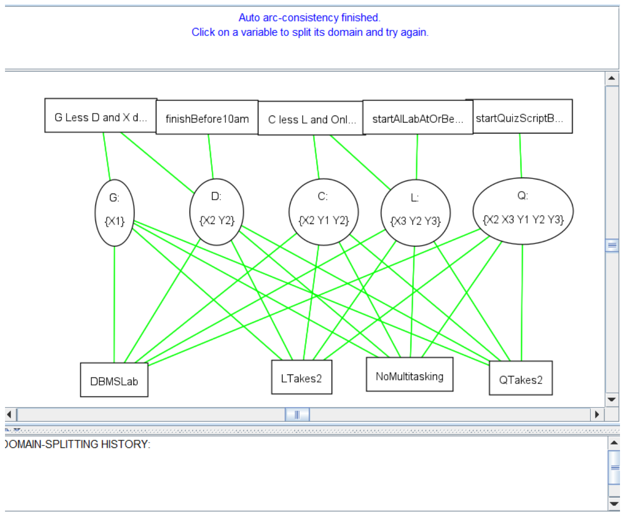
4 Task IV

In this tasks, the code generating the higher order constraints is given below, but the other constraints were simply generated by selecting the right **True False** values, using the **Custom** constraint option. Like for instance if G<D, then we ensure that the time values or the second character of each of D must be greater than G, a similar case is done for C<L; in my solution however, I merged the two constraints being the AI Lab and class has to be taken by the same person with C¡L and set the true false values for the appropriate pair, which in this case implies that while the time for C is less than that of L and L is conducted by same faculty meaning the first character has to match, only then would we say that it is true, else its false. Also for the unary constraints, we select it as per the appropriate time slots, that being the second character, if the second character is within the desired range say for instance L<3, which in turn means that only slots 1 and 2 are valid, so the potential values are Lϵ{X1,X2,Y1,Y2}. In this manner we set similar constraints for Q and D as mentioned in the analysis section.


Auto arc-consistency finished.
Click on a variable to split its domain and try again.

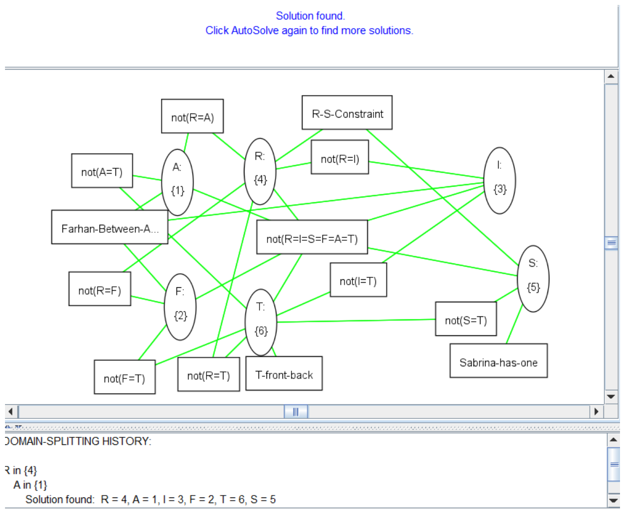
OMAIN-SPLITTING HISTORY:

Figure 5. Arc Consistency for Task III


Auto arc-consistency finished.
Click on a variable to split its domain and try again.

OMAIN-SPLITTING HISTORY:

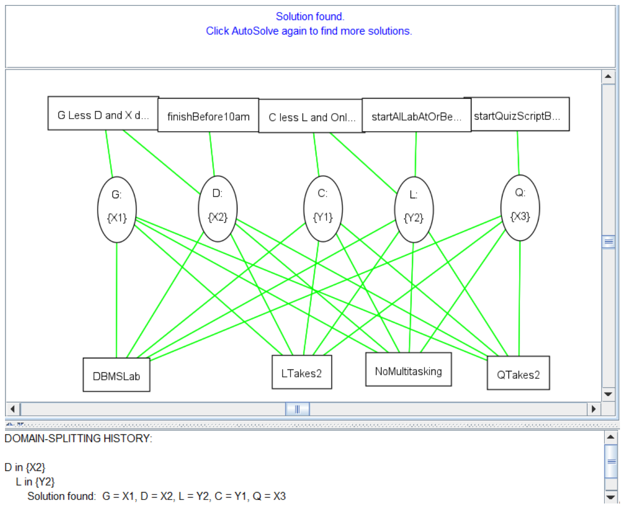
Figure 7. Arc Consistency for Task IV


Solution found.
Click AutoSolve again to find more solutions.

OMAIN-SPLITTING HISTORY:

```
 in {4}
  A in {1}
    Solution found:  R = 4, A = 1, I = 3, F = 2, T = 6, S = 5
```

Figure 6. Solution for Task III


Solution found.
Click AutoSolve again to find more solutions.

DOMAIN-SPLITTING HISTORY:

```
D in {X2}
  L in {Y2}
    Solution found:  G = X1, D = X2, L = Y2, C = Y1, Q = X3
```

Figure 8. Solution for Task IV

A DBMSLab Constraint

Firstly, when it comes to these higher-order constraints generating them manually was very time consuming but since the CSP solver makes XML files, I generated the True False combinations for these complex conditions, that I represented implicitly using C++. This applies for the other constraints mentioned below as well. I could just simply paste the True-False combinations into the XML file to get my desired combination while representing complex constraints.

For the DBMSLab Constraint, I just simply checked if either of the values for the time matches, because no two faculties, or even the same faculty can do anything else during that time

```
int main() {
    freopen("generate.txt", "w", stdout);
    vector<string> D({"X1", "X2", "X3",
    "X4", "Y1", "Y2", "Y3", "Y4"});
    vector<string> Q({"X1", "X2", "X3",
    "X4", "Y1", "Y2", "Y3", "Y4"});
    vector<string> L({"X1", "X2", "X3",
    "X4", "Y1", "Y2", "Y3", "Y4"});
    vector<string> C({"X1", "X2", "X3",
    "X4", "Y1", "Y2", "Y3", "Y4"});
    vector<string> G({"X1", "X2", "X3",
    "X4", "Y1", "Y2", "Y3", "Y4"});

    for (int i = 0;
    i < D.size(); i++) {
        for (int j = 0;
        j < Q.size(); j++) {
            for (int l = 0;
            l < L.size(); l++) {
                for (int a = 0;
                a < C.size(); a++) {
                    for (int b = 0;
                    b < G.size(); b++) {
                        if (D[i][1] == Q[j][1] ||
                        D[i][1] == L[l][1] ||
                        D[i][1] == C[a][1] ||
                        D[i][1] == G[b][1]) {
                            std::cout << "F ";
                        } else {
                            std::cout << "T ";
                        }
                    }
                }
            }
        }
    }

    return 0;
}
```

B Q Takes 2 and L Takes 2 Constraint

For the Q Takes 2 constraint, I just ensured, that the same faculty having the same character is not doing something within the next 2 hours, so if characters match meaning same faculty, their time values should have a difference greater than 2 or their time values should be less such that the task is finished before Q is started. The same logic is applicable for L.

```
int main() {
    freopen("generate2.txt", "w", stdout);
    vector<string> D({"X1", "X2", "X3",
    "X4", "Y1", "Y2", "Y3", "Y4"});
    vector<string> Q({"X1", "X2", "X3",
    "X4", "Y1", "Y2", "Y3", "Y4"});
    vector<string> L({"X1", "X2", "X3",
    "X4", "Y1", "Y2", "Y3", "Y4"});
    vector<string> C({"X1", "X2", "X3",
    "X4", "Y1", "Y2", "Y3", "Y4"});
    vector<string> G({"X1", "X2", "X3",
    "X4", "Y1", "Y2", "Y3", "Y4"});

    for (int i = 0;
    i < Q.size(); i++) {
        for (int j = 0;
        j < D.size(); j++) {
            for (int l = 0;
            l < L.size(); l++) {
                for (int a = 0;
                a < C.size(); a++) {
                    for (int b = 0;
                    b < G.size(); b++) {
                        int Qtime=Q[i][1] - '0';
                        int Qchar=Q[i][0];
                        int Ctime=C[a][1] - '0';
                        int Cchar=C[a][0];
                        int Dtime=D[j][1] - '0';
                        int Dchar=D[j][0];
                        int Gtime=G[b][1] - '0';
                        int Gchar=G[b][0];
                        int Ltime=L[l][1] - '0';
                        int Lchar=L[l][0];
                        if((Qchar==Dchar
                        && Dtime>Qtime
                        && Dtime-Qtime<2)
                         || (Qchar==Lchar
                        && Ltime>Qtime
                        && Ltime-Qtime<2)
                         || (Qchar==Cchar
                        && Ctime>Qtime
                        && Ctime-Qtime<2)
                         || (Qchar==Gchar
                        && Gtime>Qtime
                        && Gtime-Qtime<2)){
                            cout << "F ";
                        } else {
                            cout << "T ";
                        }
                    }
                }
            }
        }
    }

    return 0;
}
```

C NoMultitasking Constraint

In this case, I just checked the same faculty is not doing the another thing at the same time which in this case is done using a set as the set only stores unique elements so if the size is five it means that none of them consist of the same faculty doing the same task.

```
int main()
{
    freopen("generate3.txt", "w", stdout);
    vector<string> D({"X1", "X2",
    "X3", "X4", "Y1", "Y2", "Y3", "Y4"});
    vector<string> Q({"X1", "X2",
    "X3", "X4", "Y1", "Y2", "Y3", "Y4"});
    vector<string> L({"X1", "X2",
    "X3", "X4", "Y1", "Y2", "Y3", "Y4"});
    vector<string> C({"X1", "X2",
    "X3", "X4", "Y1", "Y2", "Y3", "Y4"});
    vector<string> G({"X1", "X2",
    "X3", "X4", "Y1", "Y2", "Y3", "Y4"});

    for (int i = 0;
    i < D.size(); i++)
    {
        for (int j = 0;
        j < Q.size(); j++)
        {
            for (int l = 0;
            l < L.size(); l++)
            {
                for (int a = 0;
                a < C.size(); a++)
                {
                    for (int b = 0;
                    b < G.size(); b++)
                    {
                        set<string> s({D[i],
                        Q[j], L[l],
                        C[a], G[b]});
                        if (s.size() == 5)
                        {
                            cout << "T-";
                        }
                        else
                        {
                            cout << "F-";
                        }
                    }
                }
            }
        }
    }

    return 0;
}
```

V Problems Faced

The primary problem would be the inability to write complex constraints. This actually only allows different types of constraints for unary and binary constraints but for higher order constraints, having to either only check the equality is not enough.

Manually setting the different True False values proves to be an arduous task. Also this solver does not allow the allDiff constraint which was discussed in class, so manually checking if all of them are different is difficult if we had to check the true false combinations upon setting a potential allDiff constraint. One thing that can however be done, is automating the process of True False generation and then manually updating the XML as I did with my code generating the True False values that I pasted to the XML file. But this is a hacky process and I would prefer there being an existing feature to carry out this task. Another option, that I followed in Task 3 is a form of mesh topology-like comparison to ensure allDiff constraint, essentially checking each variable with its neighbouring value, still its also a difficult endevour to do so.

Furthermore, many of the common shortcuts do not work, so it might seem ludicrous but I lost my work once or twice because I was using Crtl+S to save but my work kept on disappearing. Having some key shortcuts in the CSP solver would have been helpful.

Regarding the tasks themselves, I struggled a lot with mapping the appropirate constraints for the fourth task. I couldn't figure it out initially but later I had the idea of combining two different possible domains to form a new domain as discussed further in the Interesting Findings.

VI Interesting Findings

The most interesting observation would be the ability to modify XML files to automate the process of setting the values, but even more than that I think the task IV was a very interesting one and I struggled a lot with this, out of this struggle I found a way to automate the process and establish complex constraints. But the nevertheless I would say that the task IV was a very interesting task and seems like a very realistic scheduling problem with a good degree of complexity. In task 4, I first realized that we can combine 2 different aspects and even something that can be a domain itself to make something new, its like a cartesian product where for each of the courses, I can create an entirely new domain for the course variables using cartesian product of two possible domains, which is carried out here by introducing X1, X2, ..., Y1, Y2 ....