

Question 1

Introduction:

This problem states a number of food preferences between a group of people that has to be formulated into a CSP. The solution of CSP will give the possible combinations that will satisfy all the preferences stated.

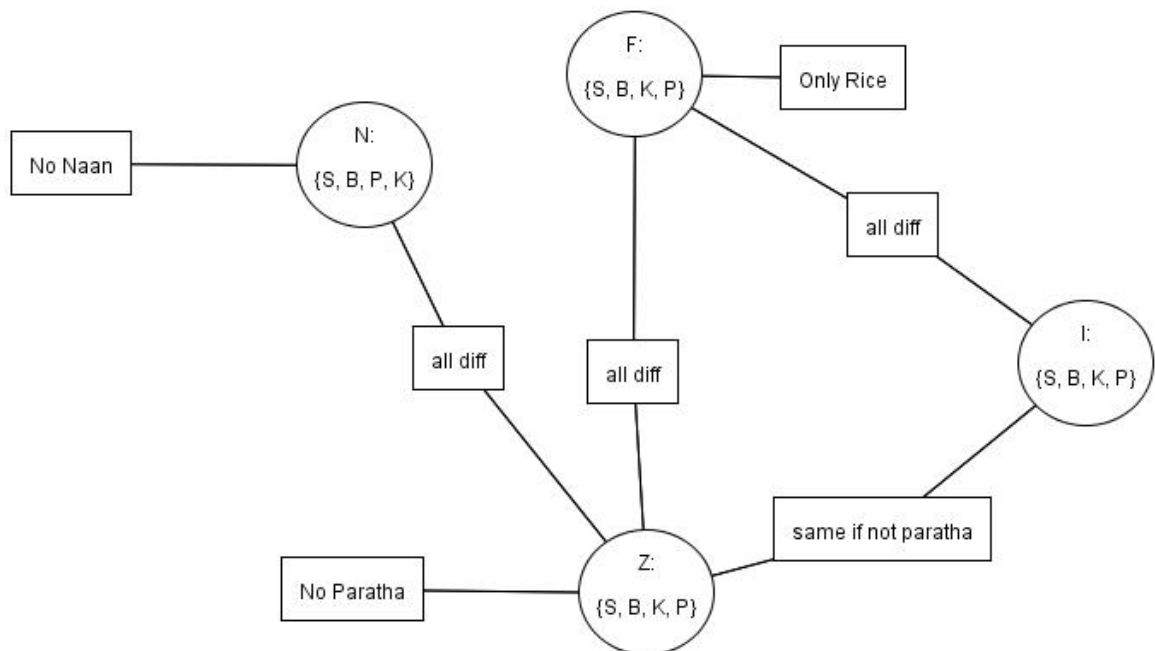
CSP Formation:

Since the problem requires assigning appropriate food preferences to the group, the food choices are the domain and each person is considered as a variable, using the first letter of their name, that has to be assigned the value.

Domain = {S, B, K, P}

Variables = {Z, I, F, N}

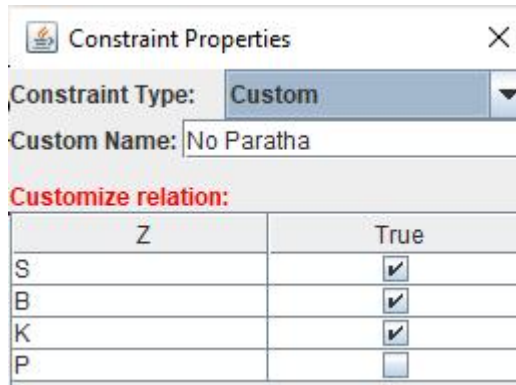
Considering the preferences stated, the resulting CSP formation is as given below. The domain values are given as string since the values to be assigned are letters.



Analysis & Explanation:

To get the corresponding CSP, the preferences are converted to constraint one by one:

The first preference can be interpreted as Zahid will never order Paratha, thus it can be converted into a unary constraint where P can not be assigned to Z since it would break the preference stated. The combination P is given as false / unchecked so as to indicate value P cannot be assigned to Z.



Z	True
S	<input checked="" type="checkbox"/>
B	<input checked="" type="checkbox"/>
K	<input checked="" type="checkbox"/>
P	<input type="checkbox"/>

The second preference concerns the variables I and F. Since they both want different dishes, an all diff constraint can be used here which essentially means any combination of values where I and F have the same values such as {SS, BB, KK, PP} are set as false. A binary constraint has to be used since we are concerned with value assignment of two variables and value assigned to one variable affects another.

F	I	True
S	S	<input type="checkbox"/>
S	B	<input checked="" type="checkbox"/>
S	K	<input checked="" type="checkbox"/>
S	P	<input checked="" type="checkbox"/>
B	S	<input checked="" type="checkbox"/>
B	B	<input type="checkbox"/>
B	K	<input checked="" type="checkbox"/>
B	P	<input checked="" type="checkbox"/>
K	S	<input checked="" type="checkbox"/>
K	B	<input checked="" type="checkbox"/>
K	K	<input type="checkbox"/>
K	P	<input checked="" type="checkbox"/>
P	S	<input checked="" type="checkbox"/>
P	B	<input checked="" type="checkbox"/>
P	K	<input checked="" type="checkbox"/>
P	P	<input type="checkbox"/>

Third preference is similar to the first preference because this also concerns the values that

can be assigned to F only. Since F will only have one of the two rice dishes, the constraint will eliminate any combination that will have other food items. This means combinations other than {S, B} has to be set to false.

F	True
S	<input checked="" type="checkbox"/>
B	<input checked="" type="checkbox"/>
K	<input type="checkbox"/>
P	<input type="checkbox"/>

The fourth preference requires a constraint between Z and other variables. Z must have different values than that of F and N and the same value assigned to I. The first part can be implemented using an all-diff between Z & F and Z & N.

Z	F	True
S	S	<input type="checkbox"/>
S	B	<input checked="" type="checkbox"/>
S	K	<input checked="" type="checkbox"/>
S	P	<input checked="" type="checkbox"/>
B	S	<input checked="" type="checkbox"/>
B	B	<input type="checkbox"/>
B	K	<input checked="" type="checkbox"/>
B	P	<input checked="" type="checkbox"/>
K	S	<input checked="" type="checkbox"/>
K	B	<input checked="" type="checkbox"/>
K	K	<input type="checkbox"/>
K	P	<input checked="" type="checkbox"/>
P	S	<input checked="" type="checkbox"/>
P	B	<input checked="" type="checkbox"/>
P	K	<input checked="" type="checkbox"/>
P	P	<input type="checkbox"/>

Z	N	True
S	S	<input type="checkbox"/>
S	B	<input checked="" type="checkbox"/>
S	P	<input checked="" type="checkbox"/>
S	K	<input checked="" type="checkbox"/>
B	S	<input checked="" type="checkbox"/>
B	B	<input type="checkbox"/>
B	P	<input checked="" type="checkbox"/>
B	K	<input checked="" type="checkbox"/>
K	S	<input checked="" type="checkbox"/>
K	B	<input checked="" type="checkbox"/>
K	P	<input checked="" type="checkbox"/>
K	K	<input type="checkbox"/>
P	S	<input checked="" type="checkbox"/>
P	B	<input checked="" type="checkbox"/>
P	P	<input type="checkbox"/>
P	K	<input checked="" type="checkbox"/>

However, the next part requires a bit of consideration due to the current implementation. If the constraint is stated as selecting the combinations containing identical values - {SS, BB, KK, PP}, this might lead to the value P being eliminated from I. This is due to the unary constraint of Z removing P from its domain and to maintain the arc consistency where $Z = I$, the value P might be removed from domain of I as well. This means the constraint has to be tweaked a little to add the cases where I might be assigned P but Z might not be able to order it since P is not within the domain so has to order something else.

Z	I	True
S	S	<input checked="" type="checkbox"/>
S	B	<input type="checkbox"/>
S	K	<input type="checkbox"/>
S	P	<input checked="" type="checkbox"/>
B	S	<input type="checkbox"/>
B	B	<input checked="" type="checkbox"/>
B	K	<input type="checkbox"/>
B	P	<input checked="" type="checkbox"/>
K	S	<input type="checkbox"/>
K	B	<input type="checkbox"/>
K	K	<input checked="" type="checkbox"/>
K	P	<input checked="" type="checkbox"/>
P	S	<input type="checkbox"/>
P	B	<input type="checkbox"/>
P	K	<input type="checkbox"/>
P	P	<input checked="" type="checkbox"/>

Combinations with I being assigned P allowed I and Z to be different.

The fifth and final preference is similar to the unary constraint applied to Z and F. Variable N will have a unary constraint where the combinations where K is assigned are set as false so N will never be assigned the value K.

N	True
S	<input checked="" type="checkbox"/>
B	<input checked="" type="checkbox"/>
P	<input checked="" type="checkbox"/>
K	<input type="checkbox"/>

This CSP produces 20 possible solutions that can be found by running AutoSolve multiple times until no more solutions are found. The constraints were checked for each solution and verified that they do not break any conditions stated.

Interesting Findings & Challenges:

Initially when the CSP was implemented under the assumption $Z = I$ was a constraint, the CSP produced 10 solutions - exactly half of what the current CSP produces. This is because in order to ensure arch consistency of $Z=I$, the CSP removes P from the domain of I, which reduces the number of possible combinations of solutions.

The constraint between Z and I was a bit difficult to understand as the combination has to be considered where I has value P assigned to it but Z can not order it because its domain

has no such value. In that case, either one of the assumption has to be made: either Z has a different value assigned in that case or I does not have the value P available in it. The first assumption is more sensible as I does not have any preference that restricts it from having P assigned. Thus the problem was solved using that assumption.

Question 2

Introduction:

This problem states the possible floors that a group of four people can be assigned to, given it follows a number of conditions. The solution requires that all people find some floor to live on and some of them may share a floor.

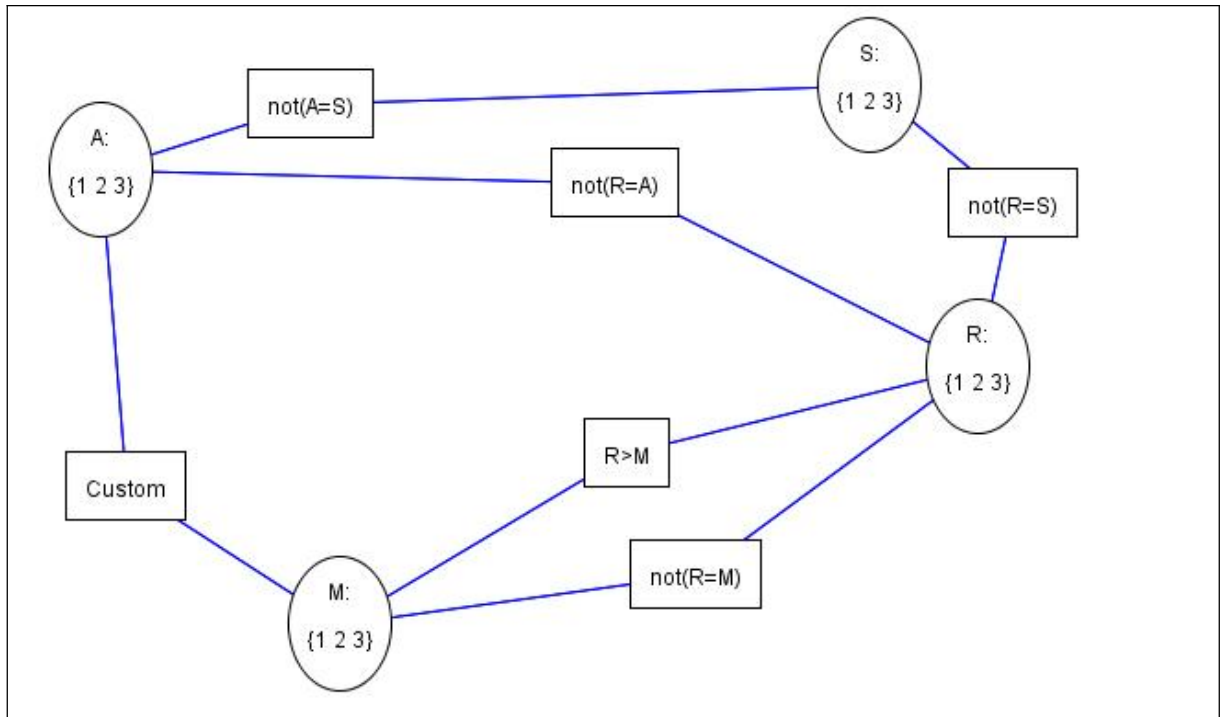
CSP Formation:

Initially, there are two ways to view the problem: either assigning the values of floors to each person (variables are people and values are the floors) or assigning a person to each floor (variables are floors and values are the people). However since multiple people can be assigned to one floor, the CSP becomes simpler with the former choice because it requires assigning one value to each variable. Thus each person is represented as a variable using the first letter of their name where the possible domain are the available floor choices.

Variables = {A, S, M, R}

Domain = {1, 2, 3}

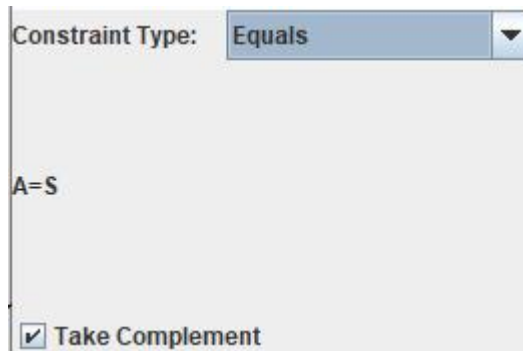
Considering the preferences stated, the resulting CSP formation is as given below. The domain values are given as integers since the values to be assigned are floor numbers.



Analysis & Explanation:

The constraints, as stated, were easier to implement using integer values.

The first preference can be ensured by a binary constraint between A and S where any combinations of them having the same value are unchecked to remove those combination from their domain. Integer constraint provides two ready made constraints that make this easy to implement - equal and complement. This removes all combinations of A and S except where they are nonidentical.



The second and third preferences must be custom constraint because constraint between A and M prevents and allows very specific constraints under some specific condition to be

accepted. In cases, when $A=M$, the they must both be assigned 2 or else the combination is unchecked. In other cases, either A or M must be assigned the value 3. Both of these were defined using one custom constraint:

A	M	True
1	1	<input type="checkbox"/>
1	2	<input type="checkbox"/>
1	3	<input checked="" type="checkbox"/>
2	1	<input type="checkbox"/>
2	2	<input checked="" type="checkbox"/>
2	3	<input checked="" type="checkbox"/>
3	1	<input checked="" type="checkbox"/>
3	2	<input checked="" type="checkbox"/>
3	3	<input type="checkbox"/>

The fourth preference involves using a binary all-diff between R and all the variables to ensure all values where R and the second variable does not have equal values. Compare to the previous question, this was easier to implement using the built in constraint setting 'complement' and 'equals'. Since number of variables and possible domain values were low as well, this did not prove to be difficult to implement.

Constraint Type: Equals

R=S

☒ Take Complement

Constraint Type: Equals

R=A

☒ Take Complement

Constraint Type: Equals

R=M

☒ Take Complement

- The final constraint requires that any value assigned to M must be lower than R and this is

also easily accomplished using 'greater than' or 'less than' constraint setting, depending on the order of the variables.



The screenshot shows a software interface for setting constraints. At the top, there is a label 'Constraint Type:' followed by a dropdown menu currently set to 'GreaterThan'. Below this, a text input field contains the expression 'R>M'. At the bottom of the interface, there is a checkbox labeled 'Take Complement' which is currently unchecked.

There are two possible solutions that are provided by the AutoSolve:

1. $S = 1, A = 2, M = 2, R = 3$
2. $S = 1, A = 3, M = 1, R = 2$

This was verified using the following logic that if R must take up an entire floor by himself and live on a higher floor than M, he can take either 2 or 3. If he takes up floor 2, then M must be on 1 in which case A must be on floor according to the third constraint. S has the choice of 2 or 1 and can only choose 1 because R has to live by himself. On the other hand, if R takes up 3, then M can be on 1 or 2. However, now neither A nor M can take floor 3 so they must be together thus M has to be on 2. S has to live on a different floor which means the only floor available for her is floor 1. No other solutions can be possible.

Interesting Findings & Challenges:

Initially my attempt was to do the task without any built in constraint functions - mainly because I was unaware of its existence since I had used the string type values in the previous question. However this led to some errors in the all diff since it required manual ticking each time for all three all diffs applied. Later on, this was solved using the built in constraint functions that make life much easier as well reduce or completely remove possibilities of error.

Question 3

Introduction:

The problem introduces a queue of six people that must be positioned in different spots and the positions must fulfill the criteria stated. For this problem, it is implicitly assumed that all six people have some position in the queue.

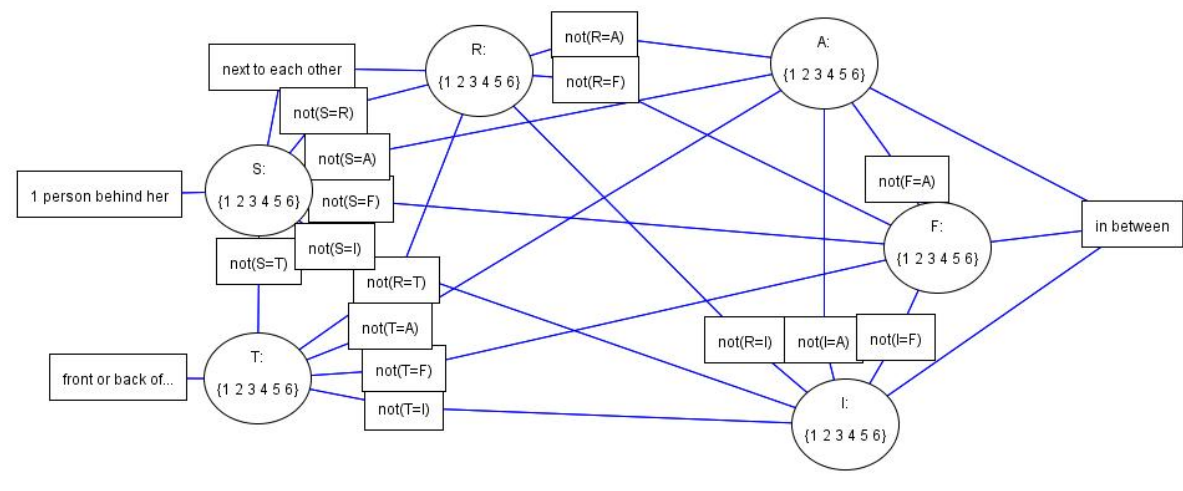
CSP Formation:

Similar to the previous question, there are two ways to view the problem: either assigning the values of spots to each person or assigning each spot with a person. In previous CSP, due to one-to-many mapping between floor to person, the floors were considered as values in domain of each person variable. This question poses no such issues since there is one-to-one mapping between a spot and a person. Either of the two ways can be correct in providing the correct solution. However, considering integers as values in domain provided a huge advantage than using strings. This is due to the multitude of options provided for setting constraints in case of integers - greater than, less than, equal to, complement etc. Thus it took less time and effort to set up all the constraints in CSP. Moreover, the chances of errors also decrease compared to setting up the custom constraints by hand.

Variables = {R, A, F, I, T, S}

Domain = {1, 2, 3, 4, 5, 6}

Considering the preferences stated, the resulting CSP formation is as given below. The domain values are given as integers since the values to be assigned are spots numbered from 1 to 6.



Analysis & Explanation:

The first condition stated in the question requires the six friends to occupy a unique spot which means the values to each variable assigned should not match any of the others. This is ensured by using a six variable all diff to ensure that all values are unique. The all-diff was instead ensured by using a binary all diff between all possible two variable combinations in the CSP, which has the same result as using a six variable all-diff. The binary all-diff was implemented using the built in constraint function 'complement' and 'equals'. Since CSP involves assigning a value to each variable, it is guaranteed that a solution will provide each friend with a spot.

(Since the number of images to be attached are too many - 19 in total - the all diff described can be found in the original CSP diagram).

The next condition states that F must be standing between A and I. This can be implemented using a three-variable constraint where F has the value A+1 and I-1 (IFA) or A-1 and I+1 (AFI). Following this, there are about 8 combinations - the first spot can be either 1, 2, 3 or 4 as there must be two consecutive spots after it, which leads to four combinations and each combination has two different orders (AFI or IFA). This was done using custom variable, where all other combinations other than these eight are selected as false.

F	I	A	True	F	I	A	True
1	6	4	<input type="checkbox"/>	3	2	1	<input type="checkbox"/>
1	6	5	<input type="checkbox"/>	3	2	2	<input type="checkbox"/>
1	6	6	<input type="checkbox"/>	3	2	3	<input type="checkbox"/>
2	1	1	<input type="checkbox"/>	3	2	4	<input checked="" type="checkbox"/>
2	1	2	<input type="checkbox"/>	3	2	5	<input type="checkbox"/>
2	1	3	<input checked="" type="checkbox"/>	3	2	6	<input type="checkbox"/>
2	1	4	<input type="checkbox"/>	3	3	1	<input type="checkbox"/>
2	1	5	<input type="checkbox"/>	3	3	2	<input type="checkbox"/>
2	1	6	<input type="checkbox"/>	3	3	3	<input type="checkbox"/>
2	2	1	<input type="checkbox"/>	3	3	4	<input type="checkbox"/>
2	2	2	<input type="checkbox"/>	3	3	5	<input type="checkbox"/>
2	2	3	<input type="checkbox"/>	3	3	6	<input type="checkbox"/>
2	2	4	<input type="checkbox"/>	3	4	1	<input type="checkbox"/>
2	2	5	<input type="checkbox"/>	3	4	2	<input checked="" type="checkbox"/>
2	2	6	<input type="checkbox"/>	3	4	3	<input type="checkbox"/>
2	3	1	<input checked="" type="checkbox"/>	3	4	4	<input type="checkbox"/>
2	3	2	<input type="checkbox"/>	3	4	5	<input type="checkbox"/>
2	3	3	<input type="checkbox"/>	3	4	6	<input type="checkbox"/>

F	I	A	True
4	3	4	<input type="checkbox"/>
4	3	5	<input checked="" type="checkbox"/>
4	3	6	<input type="checkbox"/>
4	4	1	<input type="checkbox"/>
4	4	2	<input type="checkbox"/>
4	4	3	<input type="checkbox"/>
4	4	4	<input type="checkbox"/>
4	4	5	<input type="checkbox"/>
4	4	6	<input type="checkbox"/>
4	5	1	<input type="checkbox"/>
4	5	2	<input type="checkbox"/>
4	5	3	<input checked="" type="checkbox"/>

F	I	A	True
5	4	6	<input checked="" type="checkbox"/>
5	5	1	<input type="checkbox"/>
5	5	2	<input type="checkbox"/>
5	5	3	<input type="checkbox"/>
5	5	4	<input type="checkbox"/>
5	5	5	<input type="checkbox"/>
5	5	6	<input type="checkbox"/>
5	6	1	<input type="checkbox"/>
5	6	2	<input type="checkbox"/>
5	6	3	<input type="checkbox"/>
5	6	4	<input checked="" type="checkbox"/>
5	6	5	<input type="checkbox"/>

To ensure S and R are in the consecutive position to each other, a binary constraint can be implemented. Unfortunately, there is no built in setting for the values to differ by 1. Thus the constraints had to be set by hand manually. All value combinations are set to false, except for those where values assigned to S and R differs only 1. The first variable can be in the spot 1, 2, 3, 4 or 5 since it has to have another consecutive spot beside it and the order of variables can be SR or RS since it is not specified. Thus there can be 10 such combinations.

S	R	True
1	2	<input checked="" type="checkbox"/>
1	3	<input type="checkbox"/>
1	4	<input type="checkbox"/>
1	5	<input type="checkbox"/>
1	6	<input type="checkbox"/>
2	1	<input checked="" type="checkbox"/>
2	2	<input type="checkbox"/>
2	3	<input checked="" type="checkbox"/>
2	4	<input type="checkbox"/>
2	5	<input type="checkbox"/>
2	6	<input type="checkbox"/>
3	1	<input type="checkbox"/>
3	2	<input checked="" type="checkbox"/>
3	3	<input type="checkbox"/>
3	4	<input checked="" type="checkbox"/>
3	5	<input type="checkbox"/>
3	6	<input type="checkbox"/>
4	1	<input type="checkbox"/>

4	2	<input type="checkbox"/>
4	3	<input checked="" type="checkbox"/>
4	4	<input type="checkbox"/>
4	5	<input checked="" type="checkbox"/>
4	6	<input type="checkbox"/>
5	1	<input type="checkbox"/>
5	2	<input type="checkbox"/>
5	3	<input type="checkbox"/>
5	4	<input checked="" type="checkbox"/>
5	5	<input type="checkbox"/>
5	6	<input checked="" type="checkbox"/>
6	1	<input type="checkbox"/>
6	2	<input type="checkbox"/>
6	3	<input type="checkbox"/>
6	4	<input type="checkbox"/>
6	5	<input checked="" type="checkbox"/>

The last two conditions are easy to implement using unary constraints on variables T and S. T is set to have the value 1 or 6 only and Sabrina is not allowed to be on spot 6 since she requires someone standing behind her.

S	True	T	True
1	<input checked="" type="checkbox"/>	1	<input checked="" type="checkbox"/>
2	<input checked="" type="checkbox"/>	2	<input type="checkbox"/>
3	<input checked="" type="checkbox"/>	3	<input type="checkbox"/>
4	<input checked="" type="checkbox"/>	4	<input type="checkbox"/>
5	<input checked="" type="checkbox"/>	5	<input type="checkbox"/>
6	<input type="checkbox"/>	6	<input checked="" type="checkbox"/>

Using AutoSolve provided 14 solutions for this CSP. The constraints were checked for each solution and verified that they do not break any conditions stated.

Interesting Findings & Challenges:

In this problem, I found that instead of applying all diff to all the variables at once, it works just fine if it applied to all possible binary constraints of the variables. I also discovered the magic of editing the XML file, which can save the trouble of manually clicking through the truth table one by one.

The all diff was difficult to implement for each combination of binary constraints because the manual process might be prone to error. This was solved by editing the XML file and simply copying and pasting the truth table for different combinations of the variables. In this way, the truth table needs to be implemented just once in GUI of the Applet.

When attempting to implement constraint for F between I and A, I attempted to use two constraints by manipulating integers - $I < F < A$ and $A < F < I$. However, I misjudged that it was a AND relationship and not an OR relationship. When running AutoSolve, this caused no solutions to be found since they were completely contradicting constraints.

The conditions stated were a bit confusing considering F has to be between I and A. That could either mean F could anywhere between them or they have to be beside each other. For the simplicity of the problem, I assumed the latter.

Question 4

Introduction:

The question poses a complex schedule problem of assigning two person to 5 tasks, each taking up a specified number of time slots, where all the tasks have to be completed within

8 slots of the two teachers. Aside from fulfilling the constraints, the resulting solution should have all the tasks completed within the time slots where each task should be completed only once (logical assumption based on how scheduling works.)

CSP Formation:

The problem can be approached in multiple ways. Two of the approaches that occurred to me were considering person as variables and tasks as values assigned or the opposite. Both were viable since a variable should be completed once only and each person is able to finish one task at a given time. Finally I went with the logic that made more sense sentimentally - assigning tasks to each faculty's slots.

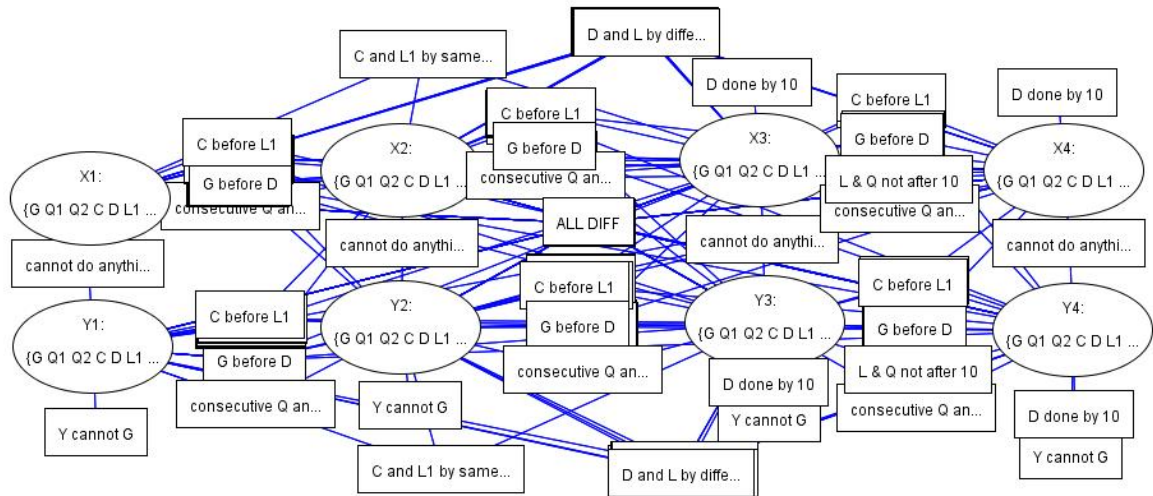
The approach selected provided me with more choices on how to choose domain values. The variables were considered to be the eight time slots of X and Y {X1, X2, X3, X4, Y1, Y2, Y3, Y4} that had to have a task assigned to them. On the other hand, values could be just the tasks itself {G, C, Q, L, D} or the tasks divided into a unit time slot so the tasks taking up multiple time slots are divided into two values {G, C, Q1, Q2, L1, L2, D}. The former has the challenge where ensuring that Q or L have occurred only twice consecutively would require three-variable constraint over all possible three consecutive variable combinations. Furthermore, ensuring that all tasks have been completed will be difficult to implement using this method. In comparison, the latter only requires binary constraint over the two consecutive variable combinations. Thus the second choice was decided to help simplify the scope of the problem.

While viewing the current CSP formation that have been decided up, it shows that there are eight variables whose values have to be assigned while there are only seven values. This required adding an eighth value '-', meaning the associated time slot has no task assigned to it. With this, the CSP was completed.

Variable = {X1, X2, X3, X4, Y1, Y2, Y3, Y4}

Values = {G, C, Q1, Q2, L1, L2, D, -}

The constraints were added, to be explained, to form the complex CSP below:



While the CSP is complex to view, it will be broken down to components based on the conditions provided in question. The domain values were strings where the names of tasks were represented using the first initial of the task.

Analysis & Explanation:

The constraints were not implemented in the order stated, but rather in the easiest to hardest/ simplest to most complex for a less painful experience.

The first requirement was fulfilled by default since each variable can have one value assigned to it in a solution.

Since I used the approach of dividing Q and L into two variables each, it requires additional constraints to ensure that Q2 always comes after Q1 and L2 always comes after L1. This is achieved using a binary constraint between consecutive time slots belonging to the same teacher. For example, if X1 had Q1, it had to have Q2 in X2. Additionally X1 and Y1 cannot have the values L2 and Q2 assigned to them. This is all achieved due to following two patterns: if there is a Q1/L1 in the first slot, no other value other than the second part of the divided task can be assigned to that time slot. In the constraints between {X1,X2} and {Y1,Y2}, I have further strengthened on this by removing the combinations where X1 and Y1 are assigned Q2 and L2. This is because without the first part of the task, the second part

obviously cannot be continued.

X1	X2	True	X1	X2	True
G	G	<input checked="" type="checkbox"/>	Q2	G	<input type="checkbox"/>
G	Q1	<input checked="" type="checkbox"/>	Q2	Q1	<input type="checkbox"/>
G	Q2	<input type="checkbox"/>	Q2	Q2	<input type="checkbox"/>
G	C	<input checked="" type="checkbox"/>	Q2	C	<input type="checkbox"/>
G	D	<input checked="" type="checkbox"/>	Q2	D	<input type="checkbox"/>
G	L1	<input checked="" type="checkbox"/>	Q2	L1	<input type="checkbox"/>
G	L2	<input type="checkbox"/>	Q2	L2	<input type="checkbox"/>
G	-	<input checked="" type="checkbox"/>	Q2	-	<input type="checkbox"/>

X1	X2	True
Q1	Q1	<input type="checkbox"/>
Q1	Q2	<input checked="" type="checkbox"/>
Q1	C	<input type="checkbox"/>
Q1	D	<input type="checkbox"/>
Q1	L1	<input type="checkbox"/>
Q1	L2	<input type="checkbox"/>
Q1	-	<input type="checkbox"/>

Next I handled the condition that L and Q should start at or before 10am. This was done by just specifying a binary constraint for $\{Y4, Y3\}$ and $\{X4, X3\}$ where the Q1 and L1 cannot start at the last time slot. This works because even if Q2 or L2 starts, this will not work due to the previous constraint where Q2 or L2 cannot occur without a prior Q1 or L1 respectively. Thus the quiz and the lab cannot start after 10, which is the last slot.

Y4	Y3	True
D	L2	<input checked="" type="checkbox"/>
D	-	<input checked="" type="checkbox"/>
L1	G	<input type="checkbox"/>
L1	Q1	<input type="checkbox"/>
L1	Q2	<input type="checkbox"/>
L1	C	<input type="checkbox"/>
L1	D	<input type="checkbox"/>
L1	L1	<input type="checkbox"/>
L1	L2	<input type="checkbox"/>
L1	-	<input type="checkbox"/>
L2	G	<input checked="" type="checkbox"/>
L2	Q1	<input checked="" type="checkbox"/>

Y4	Y3	True
G	L1	<input checked="" type="checkbox"/>
G	L2	<input checked="" type="checkbox"/>
G	-	<input checked="" type="checkbox"/>
Q1	G	<input type="checkbox"/>
Q1	Q1	<input type="checkbox"/>
Q1	Q2	<input type="checkbox"/>
Q1	C	<input type="checkbox"/>
Q1	D	<input type="checkbox"/>
Q1	L1	<input type="checkbox"/>
Q1	L2	<input type="checkbox"/>
Q1	-	<input type="checkbox"/>
Q2	G	<input checked="" type="checkbox"/>

The condition states that D has to be done by 10 am, which is the third time slot. To ensure this, the values assigned to the third and fourth slot must not contain the value D. This is implemented using a unary constraint to X3, X4, Y3, Y4 where D is not allowed as a value in their domain.

X3	True
G	<input checked="" type="checkbox"/>
Q1	<input checked="" type="checkbox"/>
Q2	<input checked="" type="checkbox"/>
C	<input checked="" type="checkbox"/>
D	<input type="checkbox"/>
L1	<input checked="" type="checkbox"/>
L2	<input checked="" type="checkbox"/>
-	<input checked="" type="checkbox"/>

Similarly it is repeated for the other variables stated.

The requirements include that G should happen before D. For this, a six variable constraint can be stated where all the combinations where D has occurred before G can be removed. However, it is a very lengthy and error prone method which is why I resorted to dividing them into sub problems and defining binary constraints. However, this requires binary constraints for all time slots except for the last time slots. For example, if X1 were to be considered, all combinations of DG are set to false for the following combination of variables {X1, X2}, {X1, X3}, {X1, X4}, {X1, Y2}, {X1, Y3}, {X1, Y4}. This is to be repeated for X2, X3, Y1, Y2 and Y3 for all the slots that come afterwards in time.

Y2	X4	True
C	L2	<input checked="" type="checkbox"/>
C	-	<input checked="" type="checkbox"/>
D	G	<input type="checkbox"/>
D	Q1	<input checked="" type="checkbox"/>
D	Q2	<input checked="" type="checkbox"/>
D	C	<input checked="" type="checkbox"/>
D	D	<input checked="" type="checkbox"/>
D	L1	<input checked="" type="checkbox"/>
D	L2	<input checked="" type="checkbox"/>
D	-	<input checked="" type="checkbox"/>

Similarly this is repeated for other combinations of variables stated.

The C must also be assigned to a earlier time slot than L or in this case, L1 since it has been divided. Previously defined constraint will ensure that L2 will also maintain this sequence, since it will only be assigned after L1. It is implemented in the same manner as the previous constraint.

Y2	X4	True
L1	G	<input checked="" type="checkbox"/>
L1	Q1	<input checked="" type="checkbox"/>
L1	Q2	<input checked="" type="checkbox"/>
L1	C	<input type="checkbox"/>
L1	D	<input checked="" type="checkbox"/>
L1	L1	<input checked="" type="checkbox"/>
L1	L2	<input checked="" type="checkbox"/>
L1	-	<input checked="" type="checkbox"/>
L2	G	<input checked="" type="checkbox"/>

This is implemented for the same combination specified as above as well.

Requirements also include that X must be assigned G and not Y. This is done by a unary constraint applied to Y1 to Y4 to exclude G from their domain.

Y1	True
G	<input type="checkbox"/>
Q1	<input checked="" type="checkbox"/>
Q2	<input checked="" type="checkbox"/>
C	<input checked="" type="checkbox"/>
D	<input checked="" type="checkbox"/>
L1	<input checked="" type="checkbox"/>
L2	<input checked="" type="checkbox"/>
-	<input checked="" type="checkbox"/>

The scheduling must also maintain that the other faculty member cannot do anything during D. This is done by binary constraints between $\{X1, Y1\}$, $\{X2, Y2\}$, $\{X3, Y3\}$, $\{X4, Y4\}$. All combinations where any variable takes the value D, the other variable cannot take any other value than - (blank).

X2	Y2	True
G	G	<input checked="" type="checkbox"/>
G	Q1	<input checked="" type="checkbox"/>
G	Q2	<input checked="" type="checkbox"/>
G	C	<input checked="" type="checkbox"/>
G	D	<input type="checkbox"/>
G	L1	<input checked="" type="checkbox"/>
G	L2	<input checked="" type="checkbox"/>

X2	Y2	True
C	L2	<input checked="" type="checkbox"/>
C	-	<input checked="" type="checkbox"/>
D	G	<input type="checkbox"/>
D	Q1	<input type="checkbox"/>
D	Q2	<input type="checkbox"/>
D	C	<input type="checkbox"/>
D	D	<input type="checkbox"/>
D	L1	<input type="checkbox"/>
D	L2	<input type="checkbox"/>
D	-	<input checked="" type="checkbox"/>

The person being assigned D cannot take L, which was ensured by binary constraints of all

possible combinations of X variables. This means that if any X variables have D assigned to it, it cannot have L1 or L2 assigned to it. This is also repeated for Y variables as well.

X3	X4	True
D	D	<input checked="" type="checkbox"/>
D	L1	<input type="checkbox"/>
D	L2	<input type="checkbox"/>
D	-	<input checked="" type="checkbox"/>
L1	G	<input checked="" type="checkbox"/>
L1	Q1	<input checked="" type="checkbox"/>
L1	Q2	<input checked="" type="checkbox"/>
L1	C	<input checked="" type="checkbox"/>
L1	D	<input type="checkbox"/>
I 1	I 1	<input checked="" type="checkbox"/>

The last requirement to be implemented was C and L being conducted by same people, which was done using three variable constraint. This was to ensure that if C occurs, then L1 must also occur. Binary constraints would not work here because it could potentially eliminate a correct scenario where C occurs in X1 and L1 occurs in X3 due to constraint between X1 and X2. This constraint has to be repeated for {X1, X2, X3} and {Y1, Y2, Y3}. If C has been assigned to X and L1 has appeared in X3 or earlier, that would ensure the constraint is satisfied and lab has been assigned to the same person being assigned C.

X3	X2	X1	True
G	G	G	<input checked="" type="checkbox"/>
G	G	Q1	<input checked="" type="checkbox"/>
G	G	Q2	<input checked="" type="checkbox"/>
G	G	C	<input type="checkbox"/>
G	G	D	<input checked="" type="checkbox"/>
G	G	L1	<input type="checkbox"/>
G	G	L2	<input checked="" type="checkbox"/>
G	G	-	<input checked="" type="checkbox"/>
G	Q1	G	<input checked="" type="checkbox"/>

If C occurs and L1 is not present, this is set as false.

X3	X2	X1	True
G	L1	Q2	<input type="checkbox"/>
G	L1	C	<input checked="" type="checkbox"/>
G	L1	D	<input type="checkbox"/>
G	L1	L1	<input type="checkbox"/>
G	L1	L2	<input type="checkbox"/>
G	L1	-	<input type="checkbox"/>
G	L2	G	<input checked="" type="checkbox"/>
G	L2	Q1	<input checked="" type="checkbox"/>
G	L2	Q2	<input checked="" type="checkbox"/>
G	L2	C	<input type="checkbox"/>

Similarly if L1 occurs without C, this is set as false.

The implicit requirement requires that the same task is not performed twice. This means that there must be an all diff constraint between the eight variables. Similar to the earlier problems solved, this constraint is divided into sub problems and implemented using binary constraints.

Solution provided by the CSP using AutoSolve was:

$X1 = G, X2 = D, X3 = Q1, X4 = Q2, Y1 = C, Y2 = -, Y3 = L1, Y4 = L2$

This was cross checked against the constraints and it was found that it is indeed the only solution to the CSP. A simple flow of logic to verify this is the fact that X must be assigned G and D must be completed within X2/Y2. This means X1 has to be G and either Y2 or X2 will be D. However, during assignment of D, the other X2/Y2 is assigned - (blank), which leaves X with 2 slots only. Since C and L must be taken by same person and requires three slots in total, Y must be the only to conduct C and L. Since C has to be conducted before L, C will be Y1 and Y3, Y4 has to be assigned L. That leaves the task of Q. Here it can only be conducted by X because Y has already all the slots filled. Furthermore, D and L must be taken by separate faculty, which disqualifies Y from taking D. Thus X2 has to be D and X3, X4 has to be Q.

Interesting Findings & Challenges:

The all diff was specifically harder for this problem. While my solution broke down the complex problem into sub problems, the sub problems itself become too much to handle at times. For the all diff, the following constraint has to be implemented 28 times to cover all scenarios.

This challenge was not exclusive to all diff because all other divided sub problems were just as huge in number and difficult to keep track of. This was partially solved using a program to print out all the necessary lines of code required in XML to duplicate a certain constraint for all its combinations.

Some of the problems were also difficult to grasp, specifically implementation of the

constraint where C and L had to be taken by same person. Although I started out with binary constraints, it was clear that it was not suitable. Due to the large number of variables and values, things became confusing at times as well. The approach I followed in order to avoid any mistakes as that would require starting from the beginning probably, I wrote down the problem and all constraints in a paper as well as ran the AutoSolve each time to ensure the correct answer was still achievable by my current implementation. Additionally, I also kept version files to ensure I could go back to a previous version which was working. These steps kept the process easy to follow and led me to complete the implementation successfully.