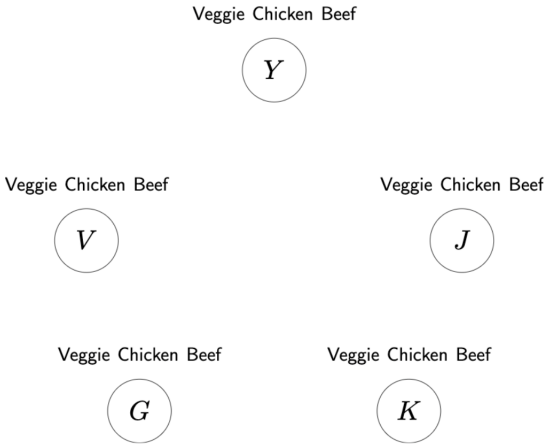
# CS221 Problem Workout Solutions

Week 6

## 1) Problem 1: Dinner Round Table CSP

(a) You and your friends (Veronica, Jarvis, Gabriela, Kanti) sit at a round table for dinner as follows, with you ($Y$) at the top, followed by Jarvis ($J$), Kanti ($K$), Gabriela ($G$), and Veronica ($V$) in clockwise order: There are three dishes on




the menu: the vegetarian deep dish pizza, the chicken quesadilla, and the beef cheeseburger, and each person plans to order exactly one dish.

Unfortunately, rather than simply enjoying the dinner, the event turns into a logistical nightmare as you and your friends impose all the following constraints upon yourselves:

i. You ($Y$) are vegetarian.
ii. If Veronica ($V$) orders beef, then Jarvis ($J$) will order veggie, and vice versa.
iii. Kanti ($K$) and Jarvis ($J$) do not want to both get non-chicken dishes.
iv. Each person wants to order something different than what the two friends sitting next to them order. For instance, Veronica ($V$) wants to order a different meal than you ($Y$) and Gabriela ($G$).

Formulating this as a constraint satisfaction problem:

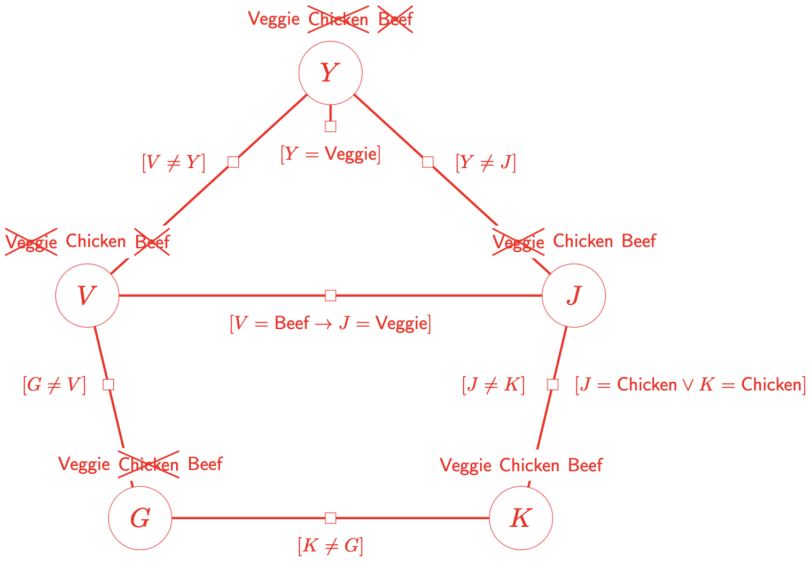
- What are the variables $X = (X_1, ..., X_n)$ based on the round table diagram above? What are the values that can be assigned to each variable?
- What are the factors between the variables? Draw edges into the round table diagram above for each factor between variables, and label each edge

with an expression that represents the constraint between the variables (e.g., $[Y = \text{Veggie}]$).

- What values are removed from the domain once arc consistency is enforced in both directions for each pair of variables? Cross out the removed values in the round table diagram above to enforce arc consistency.

**Solution** The variables are the people at the dinner, i.e. $X = [Y, J, K, G, V]$. The values that can be assigned to each variable are the possible types of dishes, i.e. [Veggie, Chicken, Beef].

Drawing and labeling the factors along edges in the diagram, and crossing out the removed values:

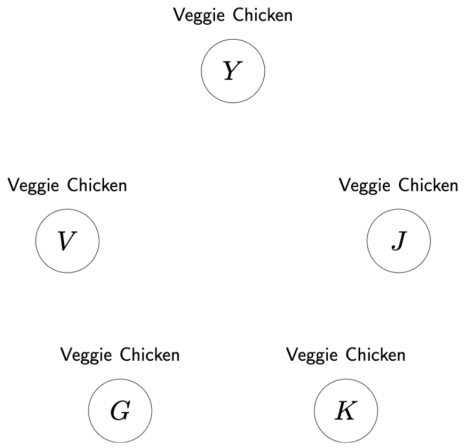



(b) Your server comes by your table and says that they are out of beef today, so you and your friends decide to rework your constraints. Now they are:

i. You ($Y$) are vegetarian.

ii. Each person wants to order something different than what the two friends sitting next to them order.

Is this new constraint problem satisfiable? For convenience, the updated table is given below.




**Solution** No, the problem is not satisfiable. No assignment will satisfy the 2nd constraint. Going clockwise: $Y$ has to order veggie, thus $J$ orders chicken, $K$ orders veggie, $G$ orders chicken, leaving $V$ to break the 2nd constraint no matter what she orders. Similar effect happens going counterclockwise.

(c) Realizing that the constraints are too strict, your group decides to relax them to the following:

i. You ($Y$) are vegetarian.

ii. Instead of hard requiring every adjacent person to order something different, it is now only a preference. To represent this, you define factors:

- $f_1(Y, J) = \mathbb{1}[Y \neq J] + 1$
- $f_2(J, K) = \mathbb{1}[J \neq K] + 1$
- $f_3(K, G) = \mathbb{1}[K \neq G] + 1$
- $f_4(G, V) = \mathbb{1}[G \neq V] + 1$
- $f_5(V, Y) = \mathbb{1}[V \neq Y] + 1$

How does defining the factors this way make the problem satisfiable? What is the maximum weight?

**Solution** Defining the factors this way makes them all strictly positive, meaning no matter the assignment, the product of all the factors will always be greater than 0, i.e.

$$\text{Weight} = \prod_{k}^{m} f_k(X) > 0$$

This by the definition of consistency and satisfiability guarantees that any assignment of values is consistent, and the problem is satisfiable.
The assignment that maximizes the weight is one where only one pair of friends shares the same order. From Part b, we know that satisfying everyone's preference of ordering a different meal from their adjacent friends is

impossible, so we have to make due as best we can. In this case, four of the five factors evaluate to 2, while one of them evaluates to 1. Thus, the weight for this assignment is

$$\text{Weight} = \prod_{k}^{m} f_k(X) = 2^4 * 1 = 16$$

Any other assignment where more than one pair of friends share the same order results in a lower weight.

## 2) Problem 2: N-Queens CSP

The N-Queens problem is a classic puzzle involving the placement of $N$ chess queens on an $N \times N$ chessboard so that no two queens threaten each other. You can find a detailed description of the 8-Queens variant and then a generalization to N-Queens here on Wikipedia if you're not too familiar with chess. As it turns out, the N-Queens problem leads itself very well into the formulation of a constraint satisfaction problem.

We'll approach the problem from the angle of analyzing the chess board one row at a time, as no two queens can be on the same row (or they threaten each other). Let our variables be the position of each queen piece $Q = Q_1, ..., Q_N$ on each row, e.g. the queen piece on row 1 has the variable $Q_1$. Let the possible values for a position be the column numbers $1, ..., N$, e.g. if we were to place the queen piece of row 1 in column N, then we would do the assignment: $Q_1 = N$.

(a) Using the variable definitions we defined above, express the constraints in this problem as factors, such that an assignment is consistent only if the following constraints of the problem are satisfied:

- No two queens are in the same column
- No two queens are on the same diagonal of the chess board

**Solution**

- "No two queens are in the same column" can be expressed as

$$f_1(Q_i, Q_j) = \mathbb{1}[Q_i \neq Q_j]$$

for all $i \neq j$.

- "No two queens are on the same diagonal of the chess board" can be expressed as

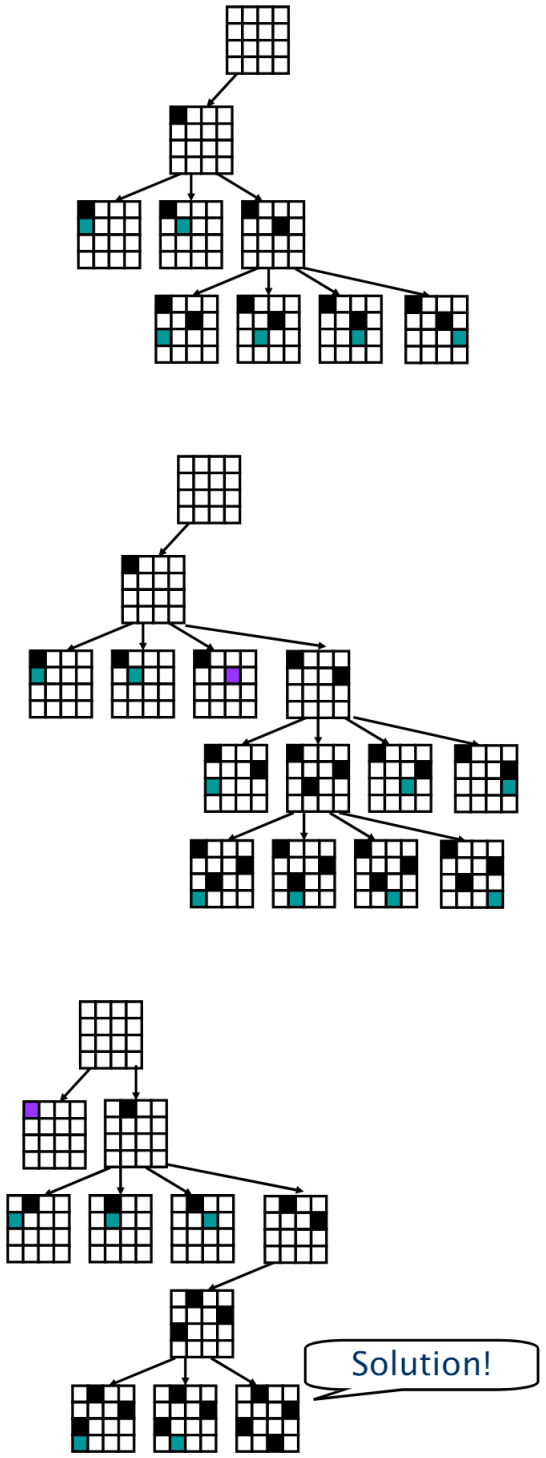
$$f_2(Q_i, Q_j) = \mathbb{1}[|Q_i - Q_j| \neq |i - j|]$$

that is, if the difference in the row placement of the two queens is equal to the difference in the column placement of the two queens, then they share a diagonal.

By using the indicator function, at least one factor will be 0 if any of the constraints are violated by an assignment. This would lead to a 0 weight, which causes the assignment to be inconsistent.

(b) Suppose $N = 4$, and we want to find a solution to the 4-Queens problem through backtracking search. We'll use the convention of starting our assignments from the top-most row 1, and go left to right from column 1 to 4, before moving on to the next row, with the last row 4 at the bottom of the board. What solution do we get for 4-queens if we were to use backtracking search with this convention? For now, just use plain backtracking; don't worry about forward checking.

**Solution** The full backtracking search is shown below. The black squares are our assignments. The blue squares are assignments that obviously violate our constraints. The purple squares represent an assignment that eventually led to dead ends later on in the branch.




Our solution ends up assigning $Q_1 = 2$, $Q_2 = 4$, $Q_3 = 1$, $Q_4 = 3$.

(c) During your backtracking search, you should've noticed that assigning $Q_1 = 1$ leads to no possible assignments for the other queens that would satisfy the problem. Repeat the backtracking search for the $Q_1 = 1$ branch, but with forward checking this time. Trace out the domain of values for our variables $Q_2, Q_3, Q_4$ at each step. You should notice that $Q_1 = 1$ is not a viable assignment sooner with forward checking than without.

Food for thought: if you had to code up backtracking search with forward checking, then what is one complication you'd have to account for with the pruned domain when going back up a branch?

**Solution** By assigning $Q_1 = 1$, we cause the following factors to equal 0:

- $f_1(Q_1 = 1, Q_2 = 1) = 0$ (column constraint)
- $f_2(Q_1 = 1, Q_2 = 2) = 0$ (diagonal constraint)
- $f_1(Q_1 = 1, Q_3 = 1) = 0$ (column constraint)
- $f_2(Q_1 = 1, Q_3 = 3) = 0$ (diagonal constraint)
- $f_1(Q_1 = 1, Q_4 = 1) = 0$ (column constraint)
- $f_2(Q_1 = 1, Q_4 = 4) = 0$ (diagonal constraint)

Hence, our remaining domain for $Q_2, Q_3$, and $Q_4$ is:

- $Q_2 : \{3, 4\}$
- $Q_3 : \{2, 4\}$
- $Q_4 : \{2, 3\}$

that is, these are the only possible assignments we can make for $Q_2, Q_3$, and $Q_4$ without violating a constraint.

Sticking to our convention of making assignments from left to right, top to bottom, we now try assigning $Q_2 = 3$ (as 3 is the left-most value remaining in the domain). Notice then for our remaining possible values for $Q_3$, we have

- $f_2(Q_2 = 3, Q_3 = 2) = 0$ (diagonal constraint)
- $f_2(Q_2 = 3, Q_3 = 4) = 0$ (diagonal constraint)

meaning the domain of $Q_3 = \{\}$ becomes the empty set. Hence we need to backtrack up the branch and try a different assignment of $Q_2 = 4$. This leads to

- $f_1(Q_2 = 4, Q_3 = 4) = 0$ (column constraint)
- $f_2(Q_2 = 4, Q_4 = 2) = 0$ (diagonal constraint)

leaving us with the domain

- $Q_3 : \{2\}$
- $Q_4 : \{3\}$

The only assignment we can do for $Q_3$ is $Q_3 = 2$, which leads to $f_2(Q_3 = 2, Q_4 = 3) = 0$, leaving the domain of $Q_4$ as the empty set. Backtracking up the branch brings us back to $Q_1$, as all possible domain options for $Q_2$ and $Q_3$ have been exhausted. Hence, there are no possible assignments that'll satisfy our problem for the branch of $Q_1 = 1$.

While this process may seem long by hand, pruning the domain via forward checking lets us make and need to check fewer assignments later on. And this can make a difference for when a computer program does the process.

Note that if we were to code this up, then we should be careful with how we prune our domain. Here, when we did the assignment of $Q_2 = 3$, we pruned the entire domain of $Q_3$, leading to the empty set. As a result, we had to go back up a branch and try a different assignment $Q_2 = 4$. But when forward checking with $Q_2 = 4$, we need to prune the domain of $Q_3 : \{2, 4\}$ from before doing the $Q_2 = 3$ assignment, not the empty set $Q_3 = \{\}$. This means that we need to restore the domain we prune when backtracking up a branch – something to keep in mind if you ever need to code this up.

**3) (optional) Problem 3: Farm Setup CSP**

Farmer Kim wants to install a set of sprinklers to water all his crops in the most cost-effective manner and has hired you as a consultant. Specifically, he has a rectangular plot of land, which is broken into $W \times H$ cells. For each cell $(i, j)$, let $C_{i,j} \in \{0, 1\}$ denote whether there are crops in that cell that need watering. In each cell $(i, j)$, he can either install ($X_{i,j} = 1$) or not install ($X_{i,j} = 0$) a sprinkler. Each sprinkler has a range of $R$, which means that any cell within Manhattan distance of $R$ gets watered. The maintenance cost of the sprinklers is the sum of the Manhattan distances from each sprinkler to his home located at $(1, 1)$. Recall that the Manhattan distance between $(a_1, b_1)$ and $(a_2, b_2)$ is $|a_1 - a_2| + |b_1 - b_2|$. Naturally, Farmer Kim wants the maintenance cost to be as small as possible given that all crops are watered. See figure below for an example.

| | | | | |
|---|---|---|---|---|
| (1,1) | (2,1) S | (3,1) C | (4,1) | (5,1) |
| (1,2) | (2,2) C | (3,2) | (4,2) S | (5,2) C |
| (1,3) | (2,3) | (3,3) | (4,3) | (5,3) |

Figure 1: An example of a farm with $W = 5$ and $H = 3$. Each cell $(i, j)$ is marked 'C' if there are crops there that need watering ($C_{i,j} = 1$). An example of a sprinkler installation is given: a cell $(i, j)$ is marked with 'S' if we are placing a sprinkler there ($X_{i,j} = 1$). Here, the sprinkler range is $R = 1$, and the cells that are shaded are the ones covered by some sprinkler. In this case, the sprinkler installation is valid (all crops are watered), and the total maintenance cost is $1 + 4 = 5$.

Farmer Kim actually took CS221 years ago, and remembered a few things. He says: "I think this should be formulated as a factor graph. The variables should be $X_{i,j} \in \{0, 1\}$ for each cell $(i, j)$. But here's where my memory gets foggy. What should the factors be?" Let $X = \{X_{i,j}\}$ denote a full assignment to all variables $X_{i,j}$. Your job is to define two types of factors:

- $f_{i,j}$: ensures any crops in $(i, j)$ are watered,
- $f_{\text{cost}}$: encodes the maintenance cost,

so that a maximum weight assignment corresponds to a valid sprinkler installation with minimum maintenance cost.

$f_{i,j}(X) =$

**Solution** For each cell $(i, j)$, let $f_{i,j}$ encode whether the crops (if they exist) in $(i, j)$ are watered:

$$f_{i,j}(X) = \left[ C_{i,j} = 0 \text{ or } \left( \min_{i',j':X_{i',j'}=1} |i' - i| + |j' - j| \right) \leq R \right]. \tag{1}$$

The first part encodes there being no crops in cell $(i, j)$. The second part encodes that if there is a crop, the closest sprinkler should be at most $R$ Manhattan distance away.

$f_{\text{cost}}(X) =$

**Solution** We define the next factor to the exponentiated negative minimum cost, so that the factor is non-negative and that maximizing the weight corresponds to minimizing the maintenance cost:

$$f_{\text{cost}}(X) = \exp\left( - \sum_{i',j':X_{i',j'}=1} |i' - 1| + |j' - 1| \right). \tag{2}$$

Any answer where the factor is non-negative and maximizing the weight corresponds to minimizing the maintenance cost is accepted. The answer should also account for the case where there are no sprinklers. Below is another acceptable answer:

$$f_{\text{cost}}(X) = \begin{cases} 1 & \text{if } X_{i',j'} = 0 \text{ for all } i', j' \\ \left( \sum_{i',j':X_{i',j'}=1} |i' - 1| + |j' - 1| \right)^{-1} & \text{otherwise} \end{cases}$$