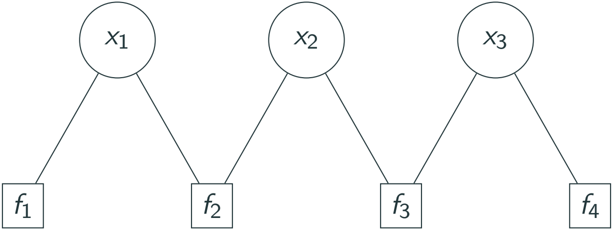
Constraint Satisfaction Problem (CSP) Review

Constraint Satisfaction Problem (CSP) Review

Defining CSPs

Variable-Based Models

Search, MDPs, and games are state-based models. CSPs are variable-based models. Think in terms of **variables** (x_i), **factors** (f_i), and **weights**.

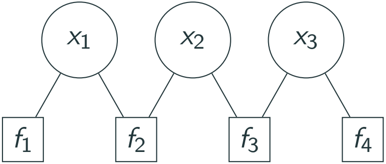


Solution to problems are assignments to variables. Use inference to make decisions (algorithms do more work now, compare to search where states did work).

Factor Graphs

Definition
Factor Graph

- Variables: $X = (X_1, \ldots, X_n)$ where $X_i \in \text{Domain}_i$
- Factors: $f_1, \ldots, f_m$, with each $f_j(X) \geq 0$.
  - How good is assignment X



- Scope of factor f_j: set of variables it depends on.
- Arity of f_j: number of variables in scope.
  - **Unary** (1 variables); **Binary** (2 variables)
- Constraints: factors that return 0 or 1.

Assignment Weights

Definition
Assignment Weight: Every assignment $x = (x_1, \dots, x_n)$ has weight

$$\text{Weight}(x) = \prod_{j=1}^{m} f_j(x)$$

- Consistent if $\text{Weight}(x) > 0$.
- A CSP is satisfiable if $\max_x \text{Weight}(x) > 0$.

Objective: Find the maximum weight assignment:

$$\text{argmax}_x \text{Weight}(x)$$

Test Your Understanding

If a CSP has an assignment $\hat{x}$ such that $\text{Weight}(\hat{x}) = 5$, is the CSP satisfiable? Recall that a CSP is satisfiable if

$$\max_x \text{Weight}(x) > 0$$

Test Your Understanding

If a CSP has an assignment $\hat{x}$ such that $\text{Weight}(\hat{x}) = 5$, is the CSP satisfiable? Recall that a CSP is satisfiable if

$$\max_x \text{Weight}(x) > 0$$

Yes, $\hat{x}$ implies that the weight of the maximizing x is greater than zero.

Test Your Understanding

If a CSP has a factor $\hat{f}(x) = 0$ for all x, is the CSP satisfiable?

Test Your Understanding

If a CSP has a factor $\hat{f}(x) = 0$ for all x, is the CSP satisfiable?

No, the weight is the product of all factors and $\hat{f}$ is always zero. Hence the weight will always be zero.

Constraint Satisfaction Problem (CSP) Review

Solving CSPs

Definition
Dependent Factors: $D(x, X_i)$ is the set of factors depending on X_i (a single variable) and x (partial assignment) but not on unassigned variables.

- If you have assigned x_1 and x_2 in x, then $D(x, X_3)$ will be all factors that depend on x_3 and one/both of x_1 and x_2.
  - i.e. if we want to assign x_3 next, what constraints (factors) are relevant?
- Idea: choose x_3 to satisfy all factors in $D(x, X_3)$!

Backtracking Search

Backtrack(x,w,Domains):

- If x is completely assigned, check if best and return.
- Choose unassigned variable X_i (component in x)
- Order Domain$_i$ (corresponds to chosen X_i)
- For each $v \in$ Domain$_i$:
  - Compute value of newly resolvable factors, setting $x_i = v$:

$$\delta = \prod_{f_j \in D(x, X_i)} f_j(x \cup \{X_i : v\})$$

  - If $\delta = 0$? Continue (at least one factor not satisfied).
  - (Optional) Shrink domain to Domains$'$ (Lookahead).
    - If any Domains$'_i$ is empty, continue.
  - Backtrack($x \cup \{X_i : v\}, w\delta$, Domains$'$)

Forward Checking (One-Step Lookahead)

- We consider an assignment for variable X_i.
- We can remove any values from neighbors of X_i that would violate factors. If any of these neighboring domains become empty, no solution, can skip this assignment.
  - Note that these 'neighbors' are only X_j that are not assigned in x.

Example: x_1, x_2, and x_3. Domain is $\{-1, 0, 1\}$. Constraints $f(x) = x_1 x_2 = -1$. See that choosing $x_1 = 0$ leads to an empty lookahead domain for x_2.

Backtracking Search

Backtrack(x,w,Domains):

- If x is completely assigned, check if best and return.
- Choose unassigned variable X_i (component in x)
- Order Domain$_i$ (corresponds to chosen X_i)
- For each $v \in$ Domain$_i$:
  - Compute value of newly resolvable factors, setting $x_i = v$:

$$\delta = \prod_{f_j \in D(x, X_i)} f_j(x \cup \{X_i : v\})$$

  - If $\delta = 0$? Continue (at least one factor not satisfied).
  - (Optional) Shrink domain to Domains$'$ (Lookahead).
    - If any Domains$'_i$ is empty, continue.
  - Backtrack($x \cup \{X_i : v\}, w\delta, \text{Domains}'$)

Variable Choices

Choosing Unassigned Variable:

- Choose the variable with the smallest domain.
- Heuristic - most constrained (smaller branching factors)

Ordering Domain$_i$:

- What order do we try values of X_i?
- Try the ones with the largest number of consistent values of neighboring variables.
- i.e. descending order of total size of possible *consistent* options for neighbors after selecting $x_i = v$.
- Impose fewest constraints on neighbors.

Arc Consistency

Definition
Arc Consistency: A variable X_i is *arc consistent* with respect to X_j if for each $x_i \in \text{Domain}_i$ there exists $x_j \in \text{Domain}_j$ such that

$$f(\{X_i : x_i, X_j : x_j\}) \neq 0$$

for all f whose scope contains X_i and X_j

If there is some choice for X_i that has no viable X_j, we don't need it! Can use this to shrink domain in lookahead.

AC-3

Algorithm: AC-3

$S \leftarrow \{X_j\}$.

While S is non-empty:

  Remove any X_j from S.

  For all neighbors X_i of X_j:

    Enforce arc consistency on X_i w.r.t. X_j.

    If Domain_i changed, add X_i to S.

Be careful, this is only a **local** view!

Beam Search

Greedy search is like DFS, but choose the assignment that gives the largest weight and explore from there.

- Will finish in $|X|$ steps.
- Cannot guarantee optimal whatsoever.
- Compromise: Greedy DFS but keep track of K best candidates at each depth.
- Still not guaranteed, but better!

Denote the size of the beam as K. Then:

- $K = 1$ is what?

Beam Search

Greedy search is like DFS, but choose the assignment that gives the largest weight and explore from there.

- Will finish in $|X|$ steps.
- Cannot guarantee optimal whatsoever.
- Compromise: Greedy DFS but keep track of K best candidates at each depth.
- Still not guaranteed, but better!

Denote the size of the beam as K. Then:

- $K = 1$ is what? Greedy
- $K = \infty$ is what?

Beam Search

Greedy search is like DFS, but choose the assignment that gives the largest weight and explore from there.

- Will finish in $|X|$ steps.
- Cannot guarantee optimal whatsoever.
- Compromise: Greedy DFS but keep track of K best candidates at each depth.
- Still not guaranteed, but better!

Denote the size of the beam as K. Then:

- $K = 1$ is what? Greedy
- $K = \infty$ is what? BFS

Beam search is like a pruned BFS. Backtracking is DFS.

Local Search

Backtracking and beam search build up assignments. Funnily enough, backtracking can't 'backtrack' information found deeper in a tree to earlier assignments. If we reach a state that would be feasible with just one variable change earlier, nothing we can do.

Solution: Local Search.

Local Search

Consider a completed assignment x. Try to improve it.

- Locality: To re-assign X_i, only need to consider factors that depend on X_i.
- Iterated Conditional Modes (ICM) For each variable, try all feasible re-assignments and pick the one with the highest weight.
- Keep looping to convergence.
- Not guaranteed optimal, local minima.
    - However, Weight(x) does monotonically increase.

Constraint Satisfaction Problem (CSP) Review

Summary

Summary

To solve CSPs we use variations of backtracking.

- Can use one-step lookahead to reduce domains after assigning a variable.
- Heuristics for choosing which variable to assign next, and what order to consider the values in the domain of that variable.
- Arc Consistency (AC-3) reduces domains to be consistent before starting the problem.
- Beam Search reduces the number of things to try in backtracking (branching) but decreases accuracy.
- Local Search given an assignment, iteratively try to improve it.

Algorithms

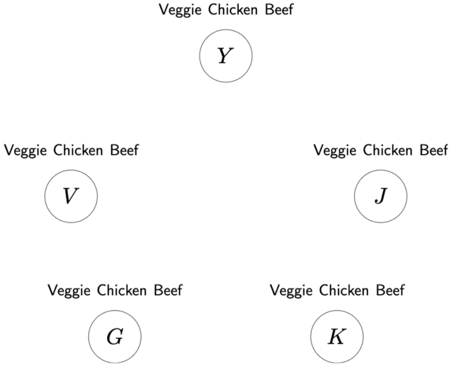
Algorithm	Strategy	Optimality	Time Complexity
Backtracking	Extend partial assignment	Exact	exponential
Beam Search	Extend partial assignment	Approximate	linear
Local Search	Modify complete assignment	Approximate	linear

Problems

Problems

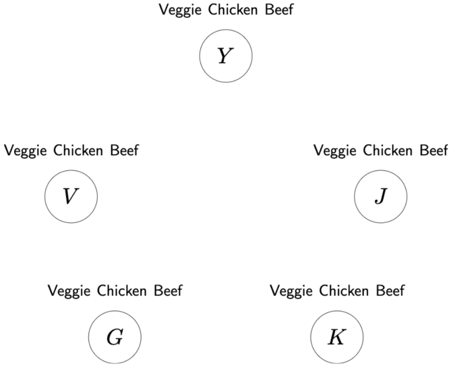
Dinner Table CSP

Dinner Table CSP



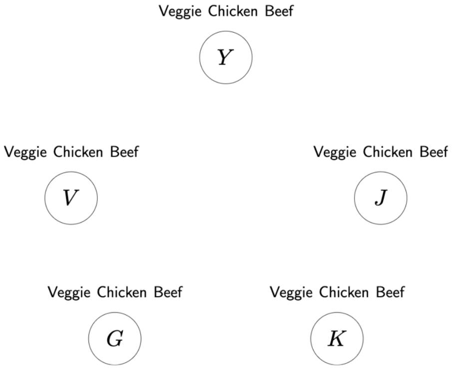
1. You (Y) are vegetarian.
2. If Veronica (V) orders beef, then Jarvis (J) will order veggie, and vice versa.
3. Kanti (K) and Jarvis (J) do not want to both get non-chicken dishes.
4. Each person wants to order something different than what the two friends sitting next to them order.

Dinner Table CSP



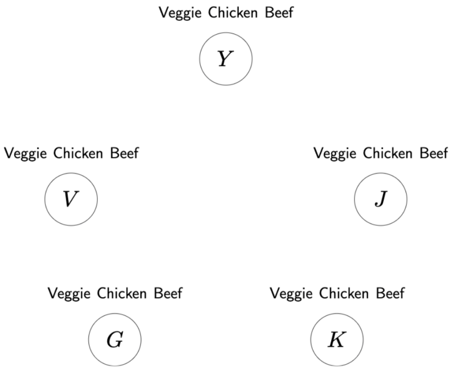
- What are the variables $X = (X_1, ..., X_n)$?
- What are the values that can be assigned to each variable?

Dinner Table CSP



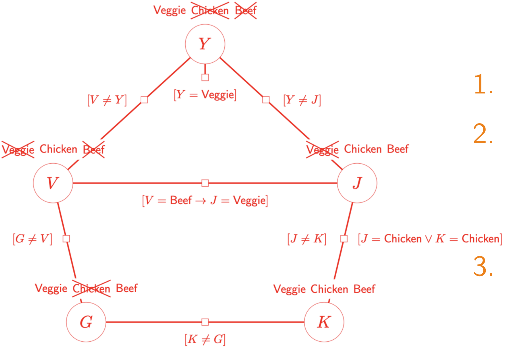
- What are the variables $X = (X_1, ..., X_n)$?
- The variables are the people at the dinner, i.e. $X = [Y, J, K, G, V]$.
- What are the values that can be assigned to each variable?
- The values that can be assigned to each variable are the possible types of dishes, i.e. [Veggie, Chicken, Beef].

Dinner Table CSP



- What are the factors between the variables?

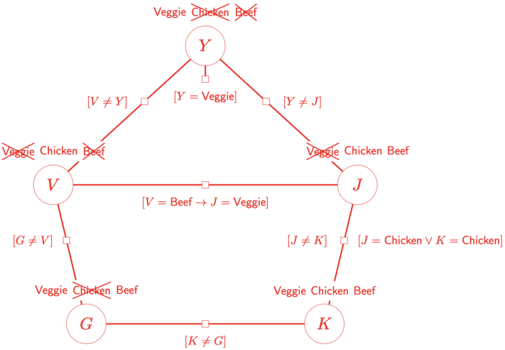
1. You (Y) are vegetarian.
2. If Veronica (V) orders beef, then Jarvis (J) will order veggie, and vice versa.
3. Kanti (K) and Jarvis (J) do not want to both get non-chicken dishes.
4. Each person wants to order something different than what the two friends sitting next to them order.



- What are the factors between the variables?

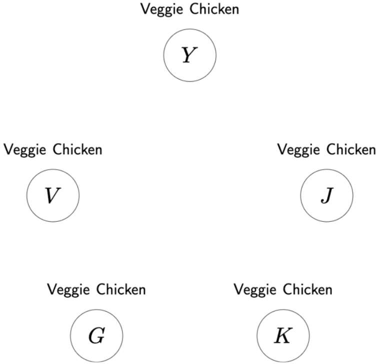
1. You (Y) are vegetarian.
2. If Veronica (V) orders beef, then Jarvis (J) will order veggie, and vice versa.
3. Kanti (K) and Jarvis (J) do not want to both get non-chicken dishes.
4. Each person wants to order something different than what the two friends sitting next to them order.

Dinner Table CSP



- Arc consistency: cross out the values in the domain that are removed by the constraints.

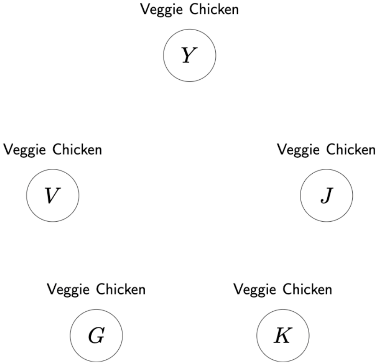
Dinner Table CSP



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:

1. You (Y) are vegetarian.
2. Each person wants to order something different than what the two friends sitting next to them order.

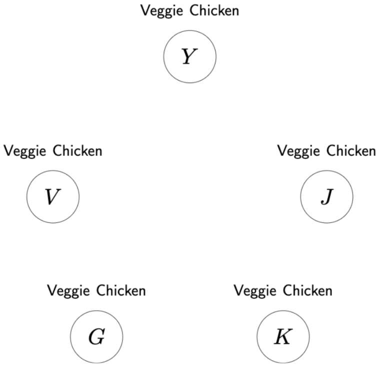
Dinner Table CSP



1. You (Y) are vegetarian.
2. Each person wants to order something different than what the two friends sitting next to them order.

Is this new constraint problem satisfiable?

Dinner Table CSP

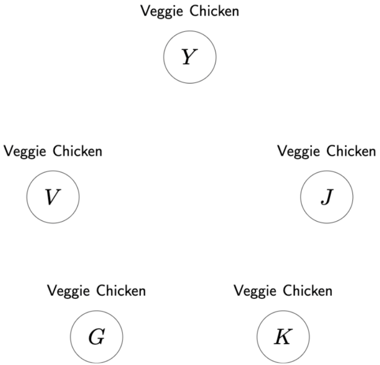


1. You (Y) are vegetarian.
2. Each person wants to order something different than what the two friends sitting next to them order.

Is this new constraint problem satisfiable?

Nope! Assign $Y =$ veggie, then alternate chicken and veggie clockwise. Y and V won't be arc consistent.

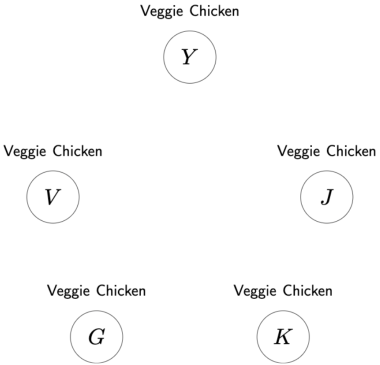
Dinner Table CSP



1. You (Y) are vegetarian.
2. Instead of hard requiring every adjacent person to order something different, it is now only a preference. To represent this, you define factors:

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

Dinner Table CSP



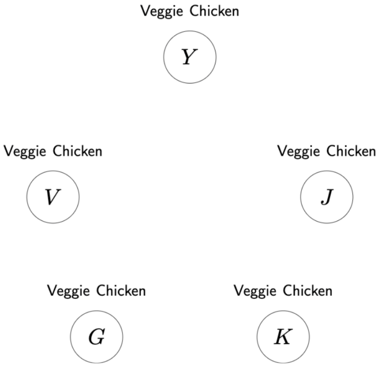
- $f_1(Y, J) = \mathbb{1}[Y \neq J] + 1$
- $f_2(J, K) = \mathbb{1}[J \neq K] + 1$
- $f_3(K, G) = \mathbb{1}[K \neq G] + 1$
- $f_4(G, V) = \mathbb{1}[G \neq V] + 1$
- $f_5(V, Y) = \mathbb{1}[V \neq Y] + 1$

$$\text{Weight}(X) = \prod_i^m f_i(X) > 0$$

An assignment to X is consistent if its weight is > 0.

A problem is satisfiable if max weight > 0.

Dinner Table CSP



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

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

What is the max weight?

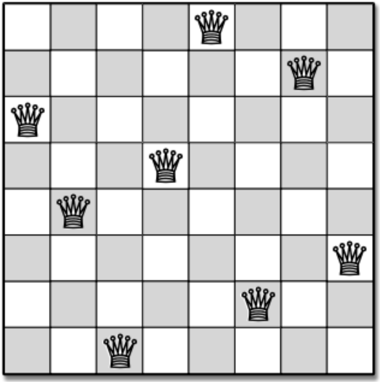
Alternate veggie, chicken clockwise starting from Y.

Weight $= 2^4 * 1 = 16$.

Problems

N-Queens

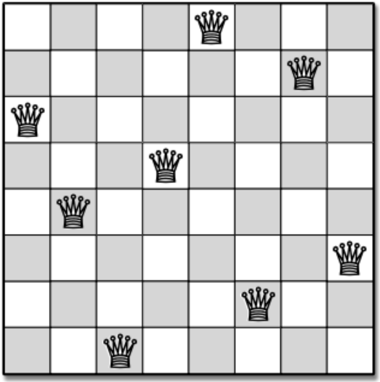
N-Queens



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.

To formulate this as a CSP, let our variables be the column placement of each queen $Q = Q_1, ..., Q_N$, where Q_1 is the queen of the first row, etc.

N-Queens

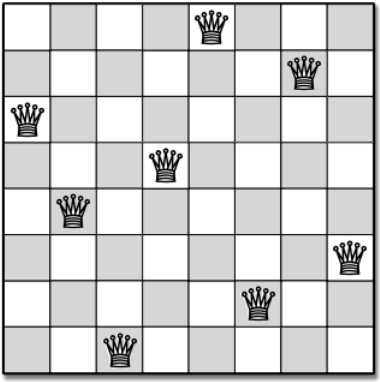


To formulate this as a CSP, let our variables be the column placement of each queen $Q = Q_1, \ldots, Q_N$, where Q_1 is the queen of the first row, etc.

Our constraints are then:

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

N-Queens

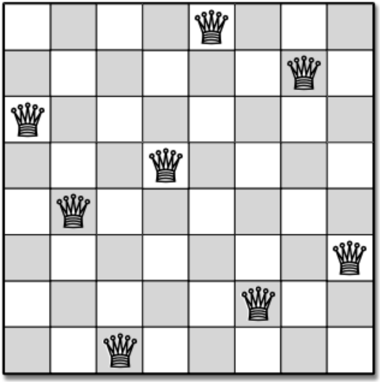


Our constraints are then:

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

How do we write these as factors?

N-Queens



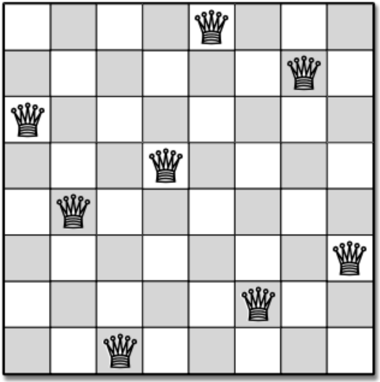
Our constraints are then:

- No two queens are in the same column

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

- No two queens are on the same diagonal of the chess board
  $f_2(Q_i, Q_j) = \mathbb{1}[|Q_i - Q_j| \neq |i - j|]$

N-Queens



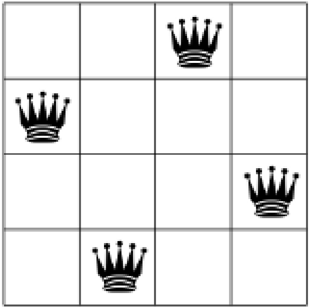
Our constraints are then:

- $f_1(Q_i, Q_j) = \mathbb{1}[Q_i \neq Q_j]$
- $f_2(Q_i, Q_j) = \mathbb{1}[|Q_i - Q_j| \neq |i - j|]$

Defining our factors this way gives us a 0 weight for an assignment that violates any constraints.

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

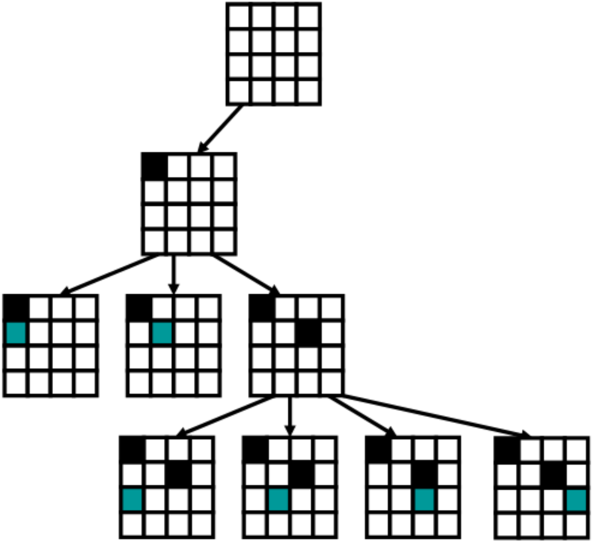
N-Queens



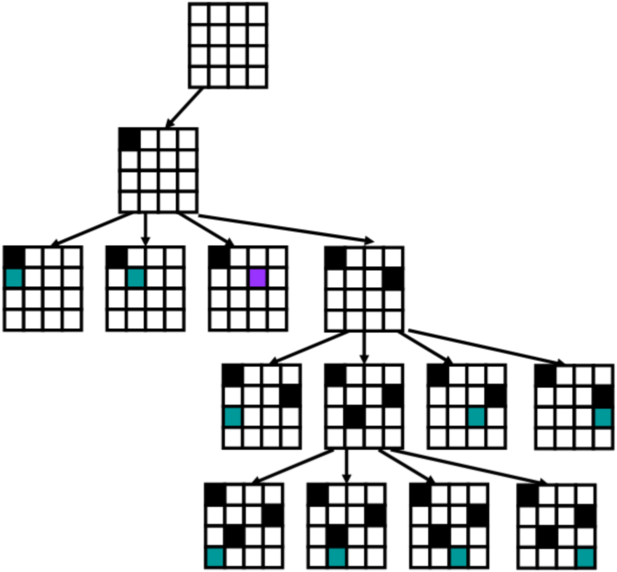
Suppose $N = 4$, and we want to find a solution to the 4-Queens problem through backtracking search.

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.

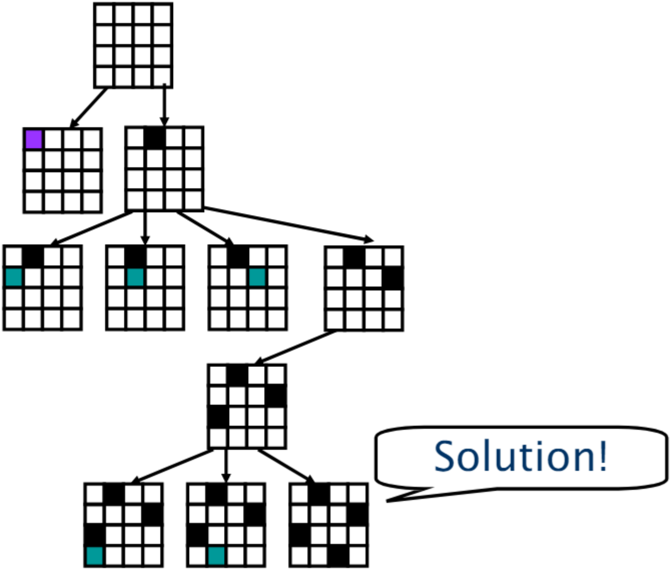
4-Queens - Backtracking Search



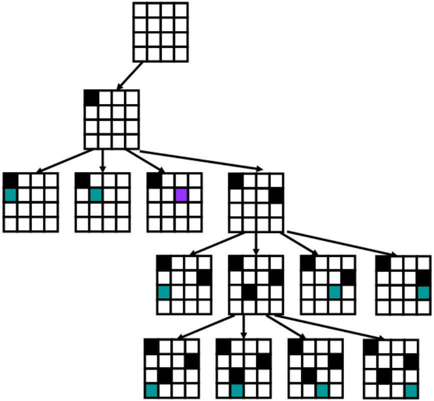
4-Queens - Backtracking Search



4-Queens - Backtracking Search



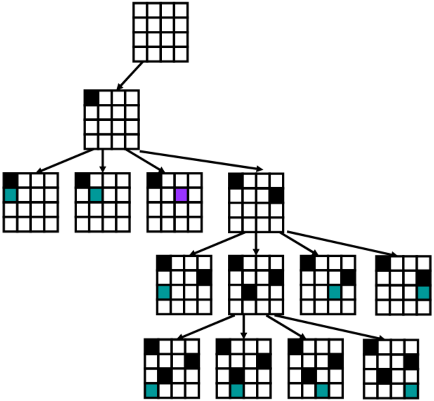
4-Queens - Forward Checking



This branch where we assign $Q_1 = 1$ ended up being a dead end.

How do we use forward checking to reduce the computation we would've needed to realize that?

4-Queens - Forward Checking



Forward checking:

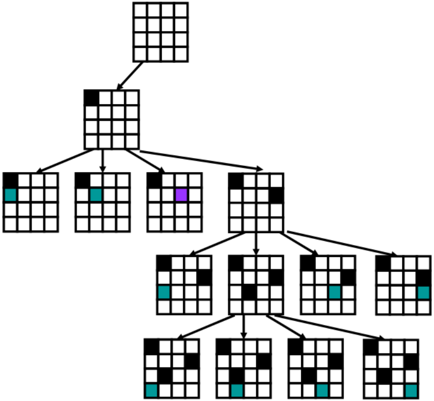
- Assign $Q_1 = 1$

Remove from the domain any values for the other variables Q_i where $f_k(Q_1, Q_i) = 0$.

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

Thus, $Q_2 : \{1, 2, 3, 4\} \rightarrow \{3, 4\}$

4-Queens - Forward Checking



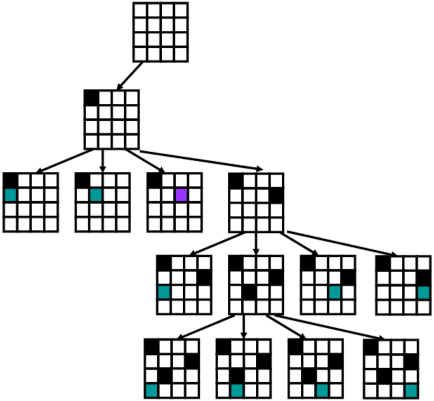
Forward checking:

- Assign $Q_1 = 1$

Remove from the domain any values for the other variables Q_i where $f_k(Q_1, Q_i) = 0$.

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

Thus, $Q_3 : \{1, 2, 3, 4\} \rightarrow \{2, 4\}$



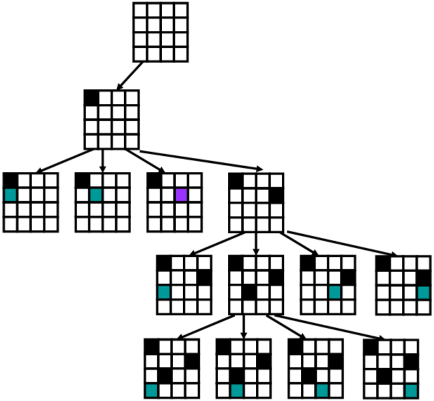
Forward checking:

- Assign $Q_1 = 1$

Remove from the domain any values for the other variables Q_i where $f_k(Q_1, Q_i) = 0$.

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

Thus, $Q_4 : \{1, 2, 3, 4\} \rightarrow \{2, 3\}$



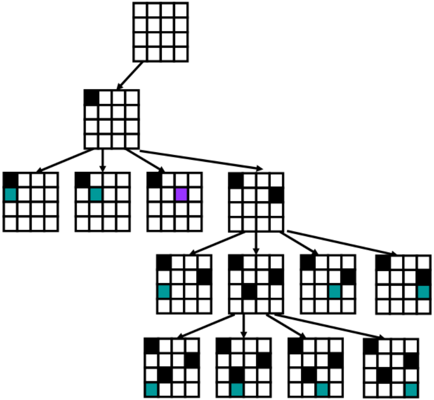
Forward checking:

- Assign $Q_1 = 1$

Remove from the domain any values for the other variables Q_i where $f_k(Q_1, Q_i) = 0$.

- $Q_2 : \{1, 2, 3, 4\} \rightarrow \{3, 4\}$
- $Q_3 : \{1, 2, 3, 4\} \rightarrow \{2, 4\}$
- $Q_4 : \{1, 2, 3, 4\} \rightarrow \{2, 3\}$

4-Queens - Forward Checking



Forward checking:

- Assign $Q_1 = 1$
- $Q_2 : \{3, 4\}$
- $Q_3 : \{2, 4\}$
- $Q_4 : \{2, 3\}$

Next, assign $Q_2 = 3$

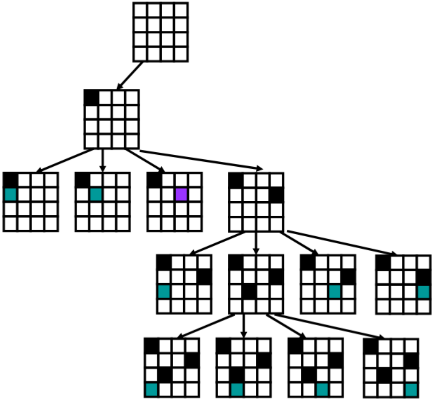
Using forward checking:

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

So Q_3 has no more values!

$Q_2 = 3$ is a dead end!

4-Queens - Forward Checking



Forward checking:

- Assign $Q_1 = 1$
- $Q_2 : \{3, 4\}$
- $Q_3 : \{2, 4\}$
- $Q_4 : \{2, 3\}$

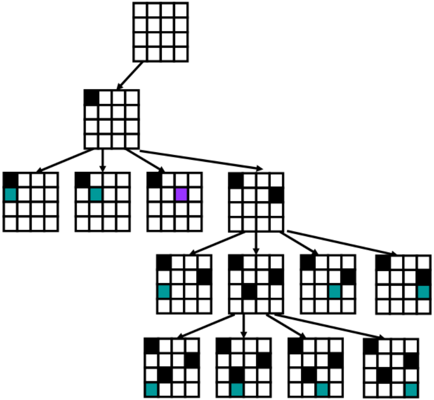
Next, assign $Q_2 = 4$

Using forward checking:

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

This leaves $Q_3 : \{2\}$ & $Q_4 : \{3\}$.

4-Queens - Forward Checking



Forward checking:

- Assign $Q_1 = 1$
- Assign $Q_2 = 4$
- $Q_3 : \{2\}$
- $Q_4 : \{3\}$

Next, assign $Q_3 = 2$

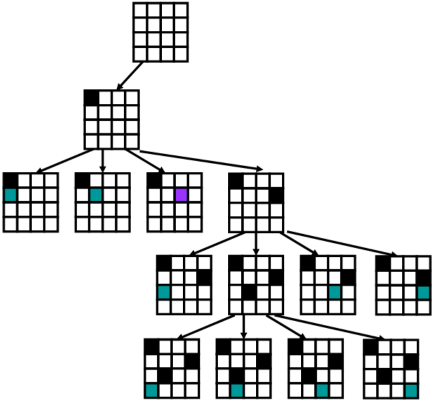
Using forward checking:

- $f_2(Q_3 = 2, Q_4 = 3) = 0$

Leaving Q_4 with no value!

Backtracking up brings us back to the Q_1 assignment, hence $Q_1 = 1$ is a dead end!

4-Queens - Forward Checking



In short, forward checking reduced our domain of possible values to assign our variables, so that later on in the branch, we do fewer assignments.

Can lead to more efficient runtimes if the forward check of all the factors isn't too intensive!

Problems

More practice - Problem 3: Farm Setup CSP