

Section 3: CSPs and Backtracking Search

*Lecturer: Stephanie Gil**Authors: Eric Helmold and Sanjana Singh***Constraint Satisfaction Problems (CSPs)**

A constraint satisfaction problem, also known as a CSP, is characterized by 3 main components outlined below:

- Decision variables denoted: $\{X_1, \dots, X_n\}$ i.e. what number of assignments need to be made? How many blanks need to be filled in? Generally the more decision variables, the more challenging the CSP.
- Domains for each variable denoted: $\{D_1, \dots, D_n\}$ i.e. what values are plausible for each decision variable. E.g. only integer values 1,2, or 3? Any real number? If the decision variables are binary the associated domains are $\{0, 1\}$.
- A set of constraints defined on subsets of variables i.e. equalities and inequalities that describe the way our assignment of values to decision variables is limited e.g. how choosing value A for X_1 might limit what we choose for X_2 if there exists a constraint relating X_1 and X_2 .

Types of Constraints

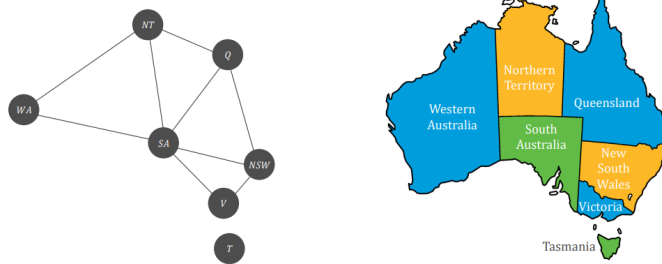
- Unary constraint - Restricts the value of a single variable e.g. $X_1 \geq 5$
- Binary constraint - Relates 2 variables to one another e.g. $X_1 + X_2 \leq 5$
- A CSP with only unary and binary constraints is called a binary CSP and can be represented using a constraint graph (see Example 1 below)
- Other constraints may involve more than 2 decision variables e.g. $X_1 + X_2 - X_3 = 12$

A solution to a CSP is a complete and consistent assignment. This means that every variable has been assigned a value and all of the constraints are satisfied.

Example 1 *Map coloring is a classic example of CSPs in action. Maps are drawn such that no 2 countries which share a border, also share the same color. Or said another way, you can easily see the borders between countries on the map because all borders have different colors on either side. Here the decision variables are the color assignments to each state/country/region of which there are N e.g. in the US there might be 50 states plus Washington DC, Puerto Rico etc. depending on the map. The domains would be the*

colors used for the map of which there may be 3 or 4 e.g. [Red, Orange, Green, Blue]. The constraints, as mentioned above, would be a set of binary constraints which stipulate that no two bordering countries can share the same color.

- Variables = {WA, NT, SA, Q, NSW, V, T}
- $D_i = \{\text{Blue, Orange, Green}\}$
- Constraints: adjacent regions have different colors

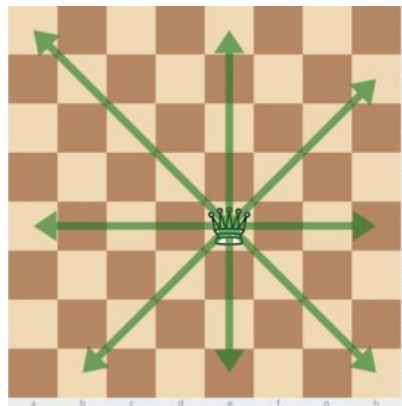


Problem 1 Consider the N -queens problem, where we wish to place N queens on a N -by- N chess board in such a configuration that no 2 queens threaten one another, i.e. in 1 move, no queen could take another queen. That is, no queen is in the same row, column, or diagonal as another queen.

Formulate this problem as a CSP. What are the decision variables? What are the variable domains? What are the constraints?

Draw a valid solution (if one exists) for $N = 1, \dots, 5$.

Hint: Recall that a queen can move any number of tiles up/down, left/right or along the diagonals going through its current square. See below for an illustration of a queen's movement on a chess board.



Source: <https://www.ichess.net>

1. We can formulate the N -queens problem as a CSP using the following 3 core requirements of a CSP:

1. Decision Variables: $\{Q_1, Q_2, \dots, Q_N\}$ Create a set of N decision variables which will denote the row assignment of each queen. Utilizing our knowledge of the problem, we know that no 2 queens can share the same column since doing so would result in their paths intersecting in that column.
2. Domains: For each variable Q_i has a domain of $\{1, 2, \dots, N\}$ since we are assigning an integer value from 1 to N representing the row assignment to each of our decision variables.
3. Constraints:
 - (a) $\forall i, j \in \{1, \dots, N\}$ for pairs i, j where $i \neq j$, set the constraint that $Q_i \neq Q_j$ to encode that no 2 queens can be placed in the same row, otherwise they would threaten one another.
 - (b) $\forall i, j \in \{1, \dots, N\}$ for pairs i, j where $i \neq j$, set the constraint that $|Q_i - Q_j| \neq |i - j|$ to encode that no 2 queens can share the same diagonal. The intuition here is that if the difference in height on the board (i.e. $|Q_i - Q_j|$) is equal to the difference in width along the board (i.e. $|i - j|$), then the slope between the 2 pieces is 1, which means they share the same diagonal.

This is just 1 of potentially many valid ways to formulate the N -queens problem as a CSP. If your approach differs, it can still be valid so long as it provides satisfactory conditions for the above 3 requirements.

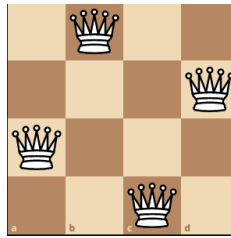
Below are examples of valid solutions for the N -queens problem for values of $N \in \{1, 2, 3, 4, 5\}$.

1. For $N = 1$ this is trivially having a queen on the board of size $[1 \times 1]$.
2. For $N = 2$ no valid solution exists. Placing 2 queens in the corners results in diagonal overlap. Placing 2 queens in the same row also creates overlap.
3. For $N = 3$ no valid solution exists. The graphic below illustrates the conflicts for any such arrangement.



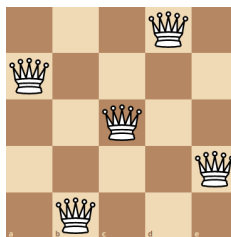
Source: <https://medium.com/@jtfeliciano/n-queens-problem-describing-the-problem-12dbd5c1508e>

4. For $N = 4$ there are 2 unique solutions and 1 fundamental solution. Below is one such example solution:



Source: <https://lichess.org/editor>

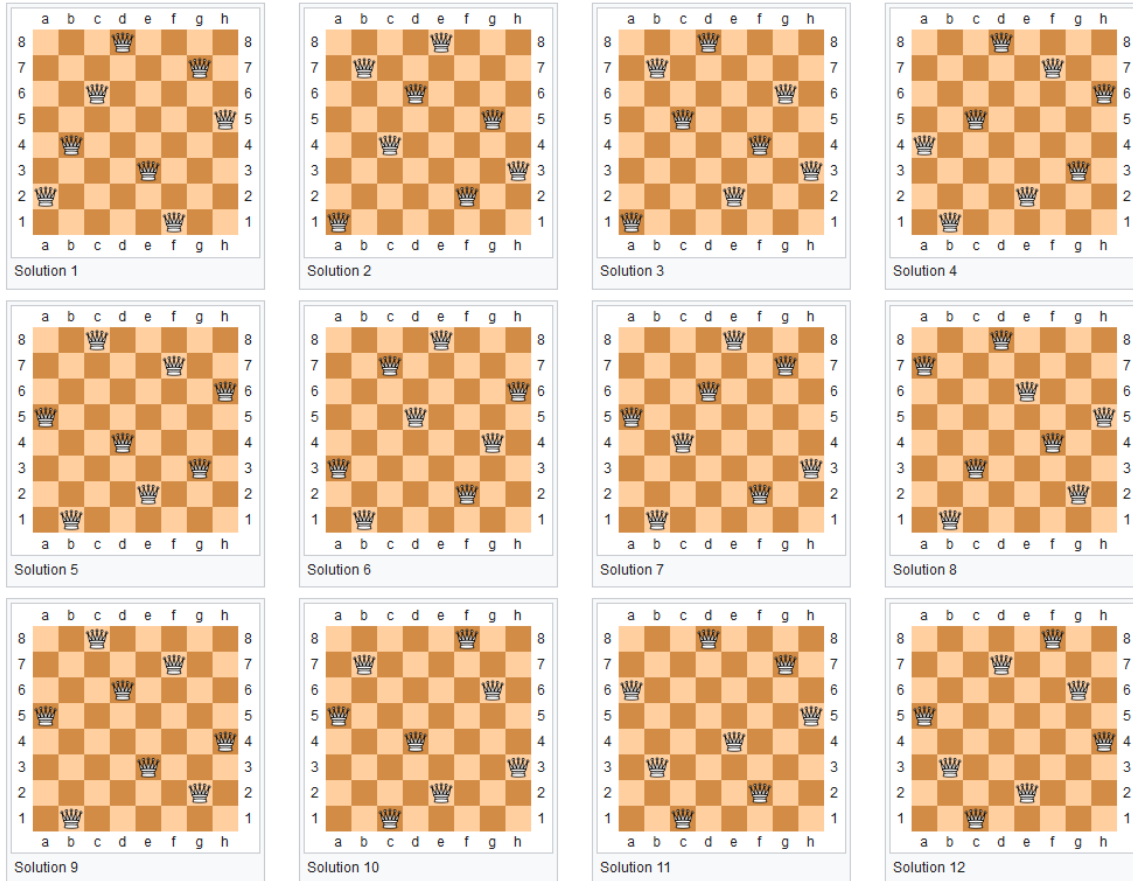
5. For $N = 5$ there are 10 unique solutions and 2 fundamental solutions. Below is one such example solution:



Source: <https://lichess.org/editor>

You can read more about the number of unique and fundamental solutions per board size here: https://en.wikipedia.org/wiki/Eight_queens_puzzle

Below are 12 fundamental solutions to the $N = 8$ queens problem (a standard chess board). Note, while there are 92 possible solutions, we tend to focus only on the 12 fundamental solutions, which are able to generate all 92 distinct board arrangements satisfying the 8-queens puzzle criteria using the symmetry operations of rotation and reflection.



Source: https://en.wikipedia.org/wiki/Eight_queens_puzzle

Problem 2 *Constraint Satisfaction Problem*

Imagine that Harvard again rewrote their CS concentration requirements. Clare just began her senior year at Harvard and wants to graduate as soon as possible. She decided to leave all of her CS requirements for her last year and all she has left to take are CS requirements. Model the problem of her trying to find a set of classes to satisfy all requirements as a CSP. What are the decision variables? What are the initial domains for each of your variables? Propose at least one selection of courses that would satisfy Clare's requirements.

Suppose the updated Harvard CS department offers the following courses:

- CS20: Logic and Discrete Math
- CS50: Intro to Programming
- CS107: Systems Development for Computational Science
- CS109a: Data Science 1: Introduction to Data Science
- CS109b: Data Science 2: Advanced Topics in Data Science
- CS124: Data Structures and Algorithms
- CS181: Machine Learning
- CS182: Artificial Intelligence
- CS184: Reinforcement Learning
- CS600: Graph Theory

In order to graduate from the CS degree program, a student must fulfill the following coursework requirements:

Algorithms Requirement: (CS50 AND CS124) OR (CS124 AND CS182) OR (CS181 AND CS600)

Machine Learning Requirement: (CS182 AND CS181) OR (CS109a AND CS109b)

Reinforcement Learning Requirement: CS181 OR CS184

Graphical Models Requirement: CS124 OR CS600

Software Development Requirement: CS107 OR CS124

Course Count Requirement: Students must take at least 8 classes in CS

Single Usage Requirement: Courses cannot be used to count toward multiple graduation requirements e.g. if you use CS182 in part to fulfill part of the algorithms requirement, then it cannot count toward either the Machine Learning Requirement.

Assume that a student can only take 4 classes per semester and that all classes are offered each semester so it does not matter which order they are taken in.

Problem and solution adapted from: https://www.cs.cmu.edu/afs/cs/academic/class/15381-s07/www/hw2/homework2_solutions.pdf

2.

We can define our decision variables to be one for each of the 8 courses Clare will take and name them according to the constraints outlined by the CS department to keep things straight and ensure all requirements have been met:

1. Algorithms Requirement 1 (AR-1)
2. Algorithms Requirement 2 (AR-2)
3. Machine Learning Requirement 1 (ML-1)
4. Machine Learning Requirement 2 (ML-2)
5. Reinforcement Learning Requirement (RL)
6. Graphical Models Requirement (GM)
7. Software Development Requirement (SD)
8. Course Count Requirement (CC)

Next we can summarize initial domains for each of these decision variables which will all be identical and encompass the full set of CS class offerings: {CS20, CS20, CS107, CS109a, CS109b, CS124, CS181, CS182, CS184, CS600}

Finally, we can finish our set up of this CSP by defining a set of constraints mathematically to describe how the decision variables relate to one another:

1. None of the decision variables can be assigned the same value i.e. only 1 class can be used to fulfill each component of each requirement: $AR-1 \neq AR-2 \neq ML-1 \neq ML-2 \neq RL \neq GM \neq SD \neq CC$
2. $AR-1, AR-2 \in \{CS50, CS124, CS182, CS181, CS600\}$
3. $ML-1, ML-2 \in \{CS182, CS181, CS109a, CS109b\}$
4. $RL \in \{CS181, CS184\}$
5. $GM \in \{CS124, CS600\}$
6. $SD \in \{CS107, CS124\}$
7. $((AR-1 == CS50) \text{ AND } (AR-2 == CS124)) \text{ OR } ((AR-1 == CS124) \text{ AND } (AR-2 == CS182)) \text{ OR } ((AR-1 == CS181) \text{ AND } (AR-2 == CS600)) = 1$
8. $((ML-1 == CS182) \text{ AND } (ML-2 == CS181)) \text{ OR } ((ML-1 == CS109a) \text{ AND } (ML-2 == CS109b)) = 1$

One possible selection of classes Clare might take to satisfy all requirements would be:

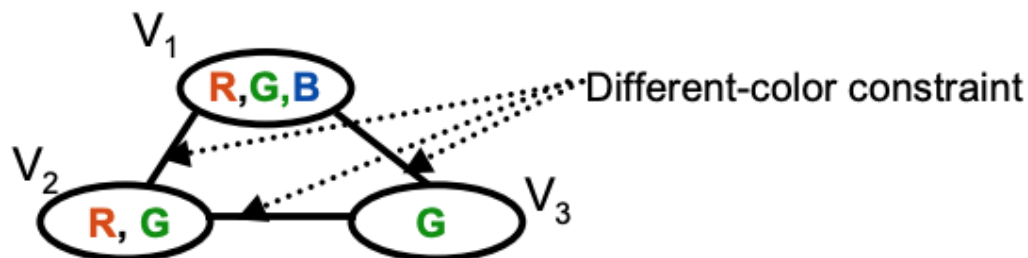
1. AR-1 = CS50
2. AR-2 = CS 124
3. ML-1 = CS109a
4. ML-2 = CS109b
5. RL = CS184
6. GM = CS600
7. SD = CS107
8. CC = CS182

Arc Consistency

Arc consistency is an important tool for reducing the possible choices we have for each decision variable and operates by logically ruling out certain assignments that would violate the constraints.

- Definition: A variable X_i is arc-consistent with respect to variable X_j if for every value in D_i there exists a value in D_j that satisfies the binary constraint on (X_i, X_j) .
- In other words, for each decision variable X_i , we should consider each value V_t remaining in its domain D_i and ask ourselves what would happen if we assigned V_t to X_i . If such an assignment is made, would there be a valid assignment for every other variable in the problem? If so, then we can move on to the next value, if not, we should remove V_t from D_i since we have learned that such an assignment is infeasible i.e. it would make us unable to satisfy one or more binary constraints.
- A CSP is said to be arc-consistent if every variable is arc-consistent with respect to every other variable.
- Enforcing arc consistency will sometimes reduce the domains to such an extent that there remains only 1 value for each decision variable, this will then have solved the CSP. Even if enforcing arc-consistency does not result in only 1 remaining value for each decision variable, it can still provide a significant reduction in the possible values to consider and can be a very useful pre-processing step for a search algorithm by considerably reducing the size of the search space.
- In certain cases, the constraints will not be restrictive enough to make arc consistency lead to any reduction in the variable domains, but we are no worse off by trying to apply it as a first step.

Problem 3 (Arc Consistency) *Is V_1 arc-consistent with V_2 and V_3 ? Is V_2 arc-consistent with V_3 ? Finally, is this CSP arc-consistent?*



Source: <https://cs.fit.edu/~dmitra/ArtInt/lectures/constraint.pdf>

3.

V_1 is not arc-consistent with V_3 since if V_1 has G assigned to it, that does not leave any legal assignments left for V_3 . V_1 is however arc-consistent with V_2 since for every value in its domain, there is a valid assignment we can make the V_2 . I.e.

1. If $V_1 = R$, then let $V_2 = G$.
2. If $V_1 = G$, then let $V_2 = R$.
3. If $V_1 = B$, then let $V_2 = R$ or G .

V_2 is not arc-consistent with V_3 since if V_2 has G assigned to it, that does not leave any legal assignments left for V_3 (same reasoning as for V_1).

V_3 however is arc-consistent with V_1 and V_2 .

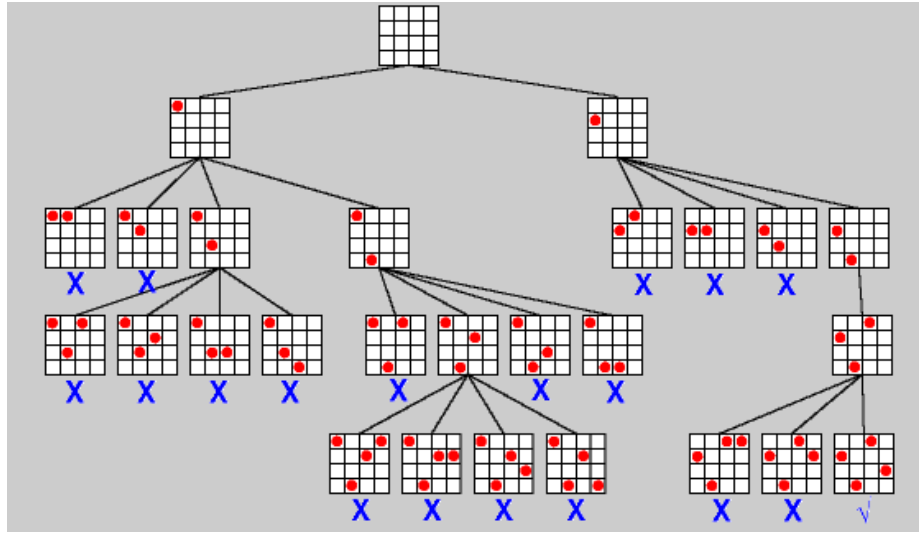
Since one or more decision variables are not arc-consistent with one another, the CSP overall is not considered to be arc-consistent.

Backtracking Search

Backtracking search is a search algorithm used to solve CSPs in a fast and efficient way, which exhaustively searches through all combinations of possible assignments to the decision variables until it finds a set of values that satisfies all constraints; however, unlike brute-force guess-and-check, backtracking search utilizes heuristics and forward-checking to greatly reduce the average search time required to locate a valid assignment.

Backtracking search is very similar to DFS, but with early pruning of search tree branches and heuristics to help decide which value assignments are most likely to lead to a valid solution. Below is a simplified conceptual outline of the steps backtracking search uses to find a solution if one exists:

1. Use a heuristic to select which of the remaining unassigned variables to assign a value to next (e.g. if we have unassigned variables A, B, and C perhaps we select decision variable C)
2. Use another heuristic to select which of the potential remaining values for the selected unassigned variable to assign to it and remove it from the set of unassigned variables (e.g. out of the options (1,2,3) assign value 3 to variable C). Also check that this is a valid assignment based on the constraints
3. Repeat steps 1 and 2 until there are no more remaining unassigned variables or we reach an impasse where we have an unassigned variable with no possible values to assign it
 - (a) If there are no remaining unassigned variables and all constraints are satisfied, then this is a valid solution and our search process ends!
 - (b) If we reach an impasse or if the assignment of values does not satisfy the constraints, then backtrack to the last assignment made, and make a different assignment and then continue again
 - (c) After each assignment, we can also use forward checking to enforce arc-consistency in order to limit the number of decision variables to consider and also check if there is a foreseeable impasse along this search branch, and if so, to backtrack early before spending unnecessary iterations to come to the same conclusion later.



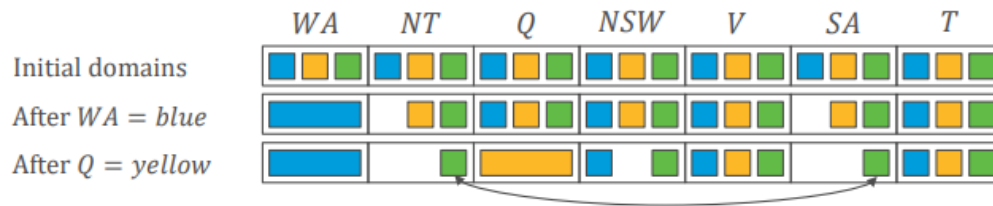
Backtracking search being used to solve the $N=4$ queens problem

Source: <https://ivoroshilin.wordpress.com/>

As mentioned above, there are 3 main components of backtracking search which differentiate it from brute force guess-and-check, they are as follows:

1. Variable ordering heuristic - This heuristic tells us which unassigned variable we should assign next in order to find a solution faster than randomly selecting.
 - (a) The minimum remaining value heuristic chooses the variable with the fewest legal values
 - (b) For example: $A : \{1, 2, 3\}, B : \{1, 2, 3, 4, 5\}, C : \{1, 8\}$. This heuristic would select variable C since it has the smallest remaining legal candidate values to try
2. Domain ordering heuristic - This heuristic tells us which value within the unassigned variable domain to try next in our iterative assignment process
 - (a) The least constraining value heuristic chooses the value that rules out the fewest choices for other variables i.e. the value that limits the domains of other decision variables based on the constraints the least
 - (b) For example: Select $C : \{1, 8\}$. This heuristic might choose 2 for instance if 8 reduced the possible candidates for all other variables the least among the options $\{1, 8\}$.
3. Forward checking - Allows us to interleave inference in our search process by enforcing arc-consistency after each variable assignment. That is, when a new assignment is made, we should go through and reduce the list of potential candidate values for the other unassigned variables based on the constraints that such an assignment would rule out.

Example 2 An example of forward checking from the map coloring CSP mentioned in lecture:



1. In the above example, we make our first assignment to WA, we assign it the color blue. This then removes blue from the domains of NT and SA by enforcing-arc consistency.
2. Then we make our next assignment of yellow to Q. This then removes yellow from the domains of NT, NSW, and SA by enforcing-arc consistency.
3. At this point, we should stop our assignment process even though we still have unassigned variables NT, NSW, and SA since we can see an obvious impasse. NT and SA both now only have green as their only remaining candidate value, which we know cannot work since these are neighboring regions and therefore cannot have the same color. Arc-consistency would fail when it examines NT and SA since there does not exist a valid assignment to SA if NT is assigned the color green. Therefore we should backtrack early and change our assignment to Q.