HW 3

# Solution 1

*Show that arc-consistency does not necessarily solve a CSP instance but just reduces the search space. When is a CSP instance 4-consistent? What is the time complexity of checking 4-consistency in a CSP instance? Are the constraints added by enforcing 4-consistency already implied by the given constraints in the CSP instance? Why don't we generally enforce 4-consistency or higher?*

## Example

Consider a CSP instance with three variables: A, B, and C, each with a domain of {1, 2, 3, 4}.

The constraints are as follows:

1. A < B
2. B < C

Initially, the domains are:

* A: {1, 2, 3, 4}
* B: {1, 2, 3, 4}
* C: {1, 2, 3, 4}

Now, let's apply AC-3:

1. Select arc (A, A < B):
   1. Prune A = 4 from domain of A
   2. Add nothing to to-do-arcs as no constraints are violated.
   3. Updated domains: A: {1, 2, 3} B: {1, 2, 3, 4} C: {1, 2, 3, 4}
2. Select arc (B, B < C):
   1. Prune B = 4 from domain of B
   2. Add (A, A < B) to to-do-arcs as domain of B changed.
   3. Updated domains: A: {1, 2, 3} B: {1, 2, 3} C: {1, 2, 3, 4}
3. Select arc (B, A < B):
   1. Prune B = 1 from domain of B
   2. Add nothing to to-do-arcs as domain of A is unchanged.
   3. Updated domains: A: {1, 2, 3} B: {2, 3, 4} C: {1, 2, 3, 4}
4. Select arc (A, A < B):
   1. Prune A = 3 from domain of A
   2. Add nothing to to-do-arcs as no constraints in B are violated.
   3. Updated domains: A: {1, 2} B: {2, 3, 4} C: {1, 2, 3, 4}
5. Select arc (C, B < C):
   1. Prune C = 1, 2 from domain of C
   2. Add (B, B < C) as domain of B unchanged.
   3. Updated domains: A: {1, 2} B: {2, 3, 4} C: {3, 4}
6. Select arc (B, B < C):
   1. Prune B = 4 from domain of B
   2. Add nothing to to-do-arcs as no constraints are violated.
   3. Updated domains: A: {1, 2} B: {2, 3} C: {3, 4}

Now, the CSP instance is in arc-consistent form, and the search space has been reduced. However, the instance is not solved yet. There are still multiple possible assignments that satisfy the constraints:

* A = 1, B = 2, C = 3
* A = 2, B = 3, C = 4

This demonstrates that arc-consistency, while powerful in reducing the search space and eliminating certain invalid assignments, does not necessarily lead to a complete solution. Additional search and inference techniques may still be required to find a valid assignment for all variables.

## 4-Consistency

* A CSP is 4-consistent if and only if for every combination of domain values to every possible distinct 3 variables there exists a common support to every other 4th variable’s domain, satisfying the constraints.
* Implication of added constraints by 4-consistency
  + Not necessarily already implied by given constraints in CSP instance.
  + Instead adds an additional constraint to ensure that every value in domain of variable has valid assignments in the domain of its neighbors.
* Enforcing 4-consistency or higher
  + Leads to more constrained problem with smaller search space.
  + Tradeoff between propagation and branching.
  + Comes with higher computation cost – time complexity increases with increase in number of levels of consistency.
  + Generally high level of consistency shall not be enforced unless problems demand it for efficient solving.
  + Lower level of consistency is preferred that strikes balance between optimizing search space and computational overhead.
* Time Complexity is O(N4D4)
  + N = number of variables
  + D = maximum domain size

# Solution 2

*Is CSP backtracking search complete or incomplete? What method is used to implement Look-Back techniques in CSP backtracking search? Pick a problem of interest in Data Science which can be solved efficiently using CSP Look-Ahead techniques. Describe the problem and the application of the CSP Look-Ahead techniques on it.*

* CSP backtracking search is complete as it does a complete assignment of variables, and it is guaranteed to find a solution if one exists.
* Conflict-directed back-jumping (CBJ) is used to implement Look-Back technique in CSP backtracking.

## N-Queens Problem

Given an N×N chessboard, the task is to place N queens on the board in a way that no two queens threaten each other. Specifically, no two queens can be in the same row, column, or diagonal.

### Algorithm

1. Variable Selection:
   * Choose a variable (queen) using a heuristic like Most Restricted Variable to place on the board.
2. Value Selection:
   * Choose a value (position) for the selected queen using a heuristic like Least Constrained Variable.
3. Forward Checking:
   * Perform forward checking to update the domains of the unplaced queens based on the current placement.
4. Constraint Propagation:
   * Apply constraint propagation techniques like Arc-Consistency to further prune the possible positions for queens.
5. Backtracking:
   * If a variable has no valid values left in its domain, backtrack to the deepest previous variable that ruled out the value, and try a different value.
6. Solution Checking:
   * Once all queens are placed, check if the placement is a valid solution. If not, continue backtracking and forward checking.

# Solution 3

*In the Dynamic Variable/Value Ordering technique, which variable do we instantiate next and in what order should the values of that variable be tried? Give an example of a problem that can be solved efficiently using Dynamic Variable/Value Ordering techniques. Draw the normal (static) search tree and the dynamically ordered search tree for that example and explain why the dynamically ordered search tree is better.*

## Variable to instantiate next?

* Minimum Remaining Value:
  + Choose the variable with the fewest legal values.
  + Choose the variable with the most constrains on the remaining variables.

## Order in which variables are to be tried?

* Least Constrained Value:
  + Choose the value the leaves most choices for the neighboring variables in the constraint graph, offering the max flexibility.

## Static vs Dynamic search tree

* In the dynamically ordered search tree, we make more informed choices.
  + By selecting columns based on the MRV heuristic, we focus on columns with fewer available positions, which are more likely to lead to a solution.
  + By using LCV within each column, we prioritize rows that limit the constraints on future placements.
* This dynamic approach prunes the search space more effectively, avoiding many unnecessary branches that would be explored in a static search tree.
* As a result, the dynamically ordered search tree is significantly more efficient in finding a solution to the **4-Queens** problem.

### Static tree

* 11
  + 21, 22 – no
  + 23
    - 3y – no
  + 24
    - 31
    - 32
      * 4y – no
    - 33, 34 – no
* 12
  + 21, 22, 23 – no
  + 24
    - 31
      * 41, 42 – no
      * 43 (solution 1)
      * 44 – no
    - 32, 33, 34 – no
* 13
  + 21
    - 31, 32, 33 – no
    - 34
      * 41 – no
      * 42 (solution 2)
      * 43, 44 – no
  + 22, 23, 24 – no
* 14
  + 21
    - 31, 32, 34 – no
    - 33
      * 4y – no
  + 22
    - 3y – no
  + 23, 24 – no

### Dynamic tree

* 12
  + 24
    - 31
      * 43 (solution)

We start with column 2 based on MRV and select row 1 based on LCV, repeat for every iteration and it directly leads to the solution.

# Solution 4

*What are Simple Temporal Problems (STPs)? How can they be solved efficiently? What are the earliest and latest schedules for a consistent STP instance? What are Disjunctive Temporal Problems (DTPs)? Where do they arise? What is the complexity of solving them? How can they be solved efficiently in practice?*

## Simple Temporal Problems (STPs)

* STPs are a class of scheduling problems where a set of tasks need to be executed subject to temporal constraints. Each task has a duration and a time window within which it can start and end.
* They can be solved efficiently using,
  + Bellmann Ford algorithm
    - If the problem, can be converted into a distance graph.
    - Then the shortest path computation with negative costs can give an optimal solution.
    - Consistency of this temporal problem then corresponds to absence of negative cost cycle in the distance graph.
  + Time-Indexed Formulation
  + Floyd Warshall's algorithm – can also be used with distance graph for shortest path computation.
  + Simple Temporal Problem in Disjunctive Scheduling (STPDS) algorithm
* Schedules for consistent STP instance
  + Earliest:
    - minimizes the overall completion time of all tasks while respecting all constraints.
  + Latest:
    - allows for maximum flexibility in task execution, ensuring that no constraint is violated.
  + Where?

## Disjunctive Temporal Problems (DTPs)

* DTPs extend the concept of STPs by introducing disjunctive constraints.
* In DTPs, tasks can be executed in one of several alternative ways, and these alternatives are subject to temporal constraints.
* DTPs are encountered in various scenarios where scheduling, has more constraints attached to it like resource allocation, starting and ending process, etc. where tasks have multiple modes of execution.
* Solving DTPs is known to be NP-hard.
  + Finding an optimal solution can be computationally intensive.
  + Number of possible schedules and combinations of disjunctive constraints can grow exponentially with the size of the problem.
  + Presence of disjunctive constraints adds complexity to the problem.

## Solving STPs and DTPs

* Can be converted to CSP and be represented as a graph.
  + Variables:
    - Each task with its associated modes becomes a variable in the CSP.
    - The domain of each variable is the set of modes.
  + Constraints:
    - Duration and mode selection constraints are directly translated into unary constraints on the variables.
    - Concurrent tasks and temporal constraints are translated into binary constraints between the variables.
    - Slack variables can be included as additional variables with their respective domains and constraints.
      * For each constraint in the form of **end\_time - start\_time <= duration**, a slack variable is introduced to represent the difference between the duration and the actual time taken.
* Graph representation
  + Nodes:
    - Each variable in the CSP corresponds to a node in the distance graph.
    - The nodes represent the possible choices for each task.
  + Edges:
    - Edges between nodes are added to represent constraints.
    - In the context of DTP, these constraints may be temporal constraints, mode selection constraints, or concurrent task constraints.
    - The weights on the edges represent the "distance" or cost associated with transitioning from one choice to another.
* After conversion to a distance graph and cost matrix can be constructed.
  + Check for -ve cost cycles for consistency.
  + Apply Bellman Ford or Floyd Warshall’s algorithm to find a solution.
  + For more complex where directed distance graph representation is not feasible more specialized algorithms may be used.

# Solution 5

*What are Bayesian Networks (BNs)? What are the various kinds of queries that can be formulated on a BN? Compare CSPs and BNs with respect to their representational power, the queries they support, the techniques used for answering the queries, and the efficiency of doing so.*

* Bayesian Networks (BNs) are graphical models that represent probabilistic relationships among a set of variables.
* Each node is associated with a conditional probability distribution that quantifies the likelihood of its value given the values of its parents.
* Defined by a directed acyclic graph (DAG) where:
  + Nodes represent random variables.
  + Edges represent probabilistic dependencies.

## Types of Queries

* Distribution Inference Queries
  + Query that involves inferring or computing a probability distribution over the variables in the Bayesian Network.
  + It can encompass both MAP and Marginal Probability queries.
* Map Queries
  + Seeks to find the most probable assignment of values to a set of variables given evidence.
  + Finds the assignment that maximizes the posterior probability.
* Marginal Map Queries (MMAP)
  + Involves computing the probability distribution of a subset of variables in the network, considering all possible values of the remaining variables.
* Probability Queries
  + Given evidence about some variables, compute the probability of other variables or events occurring.

## Comparison

|  |  |  |
| --- | --- | --- |
|  | CSPs | BNs |
| Representational Power | deterministic constraints and logical relationships among variables  *O* (*n* + *m*), where *n* is the number of variables, and *m* is the number of constraints | probabilistic relationships and dependencies among variables  *O* (*n* + *m*), where *n* is the number of nodes (variables), and *m* is the number of edges |
| Queries Supported | Support:   * satisfiability * solution finding * optimization queries   related to deterministic constraints | Support:   * probability queries * distribution inference queries * MAP queries * Marginal MAP queries |
| Techniques for answering queries | * Constraint Propagation – Arc Consistency * Backtracking – Conflict directed back-jumping. * Look Ahead – Forward checking. * Dynamic variable/value ordering | * Variable Elimination |
| Computational Efficiency | Arc Consistency – O(NkDk) where N=number of variables, D=domain size and k=number of consistent variables  CBJ - *O*(*bd*), where *b* is the branching factor and *d* is the depth of the search tree.  FC – depends on tree structure. In worst case exponential time  DVO – done heuristically where it needs to find variables and values using MRV and LCV at every new branch explored | Depends on the structure of the Bayesian Network and the complexity of the conditional probability distributions.  It can have exponential time complexity in the worst case. |