COMP3702 Tutorial 4

Aryaman Sharma

August 2022

Constraint Satisfaction Problem (CSP)

Definition (Costraint Satisfaction Problem)

A CSP is given by:

Constraint Satisfaction Problem (CSP)

Definition (Costraint Satisfaction Problem)

A CSP is given by:

- A set of variable $V_1, V_2, ..., V_n$.

Constraint Satisfaction Problem (CSP)

Definition (Costraint Satisfaction Problem)

A CSP is given by:

- A set of variable $V_1, V_2, ..., V_n$.
- Each variable V_i has an associated domain of possible values.

Constraint Satisfaction Problem (CSP)

Definition (Costraint Satisfaction Problem)

A CSP is given by:

- A set of variable $V_1, V_2, ..., V_n$.
- Each variable V_i has an associated domain of possible values.
- A set of constraints on various subsets of the variables which are logical predicates specifying legal combinations of values for these variables.

Constraint Satisfaction Problem (CSP)

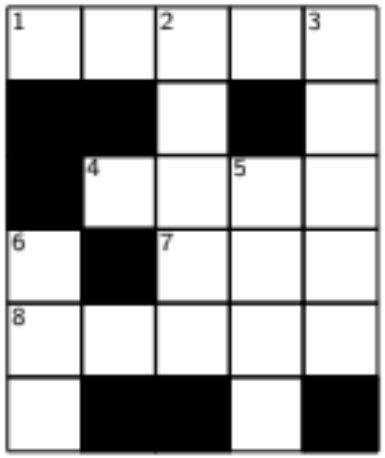
Definition (Costraint Satisfaction Problem)

A CSP is given by:

- A set of variable $V_1, V_2, ..., V_n$.
- Each variable V_i has an associated domain of possible values.
- A set of constraints on various subsets of the variables which are logical predicates specifying legal combinations of values for these variables.
- A **Model** of a CSP is an assignment of values variables that satisfies all of the constraints.

4.1

A **crossword puzzle** can be modeled as a CSP. Consider the puzzle below:



Words: AFT, ALE, EEL, HEEL, HIKE, HOSES, KEEL, KNOT, LASER, LEE, LINE, SAILS, SHEET, STEER, TIE.

Words must start at a position label number and run left-to-right or top-to-bottom only. Intersection boxes can contain only one letter. Each word in the list provided can be used only once. The words don't overlap, only intersect, so, e.g. 2 is 2-down only, and 7 is 7-across only (nb. this isn't a constraint, this is part of a typical crossword problem definition).

Exercise 4.1.

a) List the variables, and their domains, in the CSP representation of this crossword puzzle.

b) List the constraints in the CSP representation of the crossword puzzle.

4.1 a

Variables

4.1 a

Variables

- 1-across

4.1 a

Variables

- 1-across
- 2-down, 3-down, 4-across, 5-down, 6-down, 7-across and 8-across.

4.1 a

Variables

- 1-across
- 2-down, 3-down, 4-across, 5-down, 6-down, 7-across and 8-across.

Domains

4.1 a

Variables

- 1-across
- 2-down, 3-down, 4-across, 5-down, 6-down, 7-across and 8-across.

Domains

- For all variables: {AFT, ALE, ..., TIE }

4.1 b

List **constraints** in this CSP

4.1 b

List **constraints** in this CSP

- **Domain constraint** Variable can only be set to words of correct length.

4.1 b

List **constraints** in this CSP

- **Domain constraint** Variable can only be set to words of correct length.
- **Binary constraint** Letters used at intersection must be equal.

4.1 b

List **constraints** in this CSP

- **Domain constraint** Variable can only be set to words of correct length.
- **Binary constraint** Letters used at intersection must be equal.
- **Global constraint** Each word is used only once.

Constraint graph: A bipartite graph

Constraint graph: A bipartite graph

- There is a circle or oval-shaped node for each variable.

Constraint graph: A bipartite graph

- There is a circle or oval-shaped node for each variable.
- There is a rectangular node for each constraint.

Constraint graph: A bipartite graph

- There is a circle or oval-shaped node for each variable.
- There is a rectangular node for each constraint.
- There is a domain of values associated with each variable node.

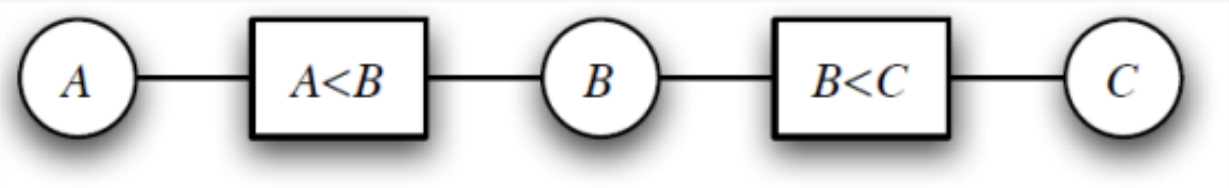
Constraint graph: A bipartite graph

- There is a circle or oval-shaped node for each variable.
- There is a rectangular node for each constraint.
- There is a domain of values associated with each variable node.
- There is an arc from variable X to each constraint that involves X.

Constraint graph: A bipartite graph

- There is a circle or oval-shaped node for each variable.
- There is a rectangular node for each constraint.
- There is a domain of values associated with each variable node.
- There is an arc from variable X to each constraint that involves X.

Example *binary* constraint network:



Variables: $\{A, B, C\}$, Domains: (not specified)
Constraints: $r_1(A, B) = (A < B)$, $r_2(B, C) = (B < C)$
Arcs: $\langle A, r_1(A, B)\rangle$, $\langle B, r_1(A, B)\rangle$,...

4.2

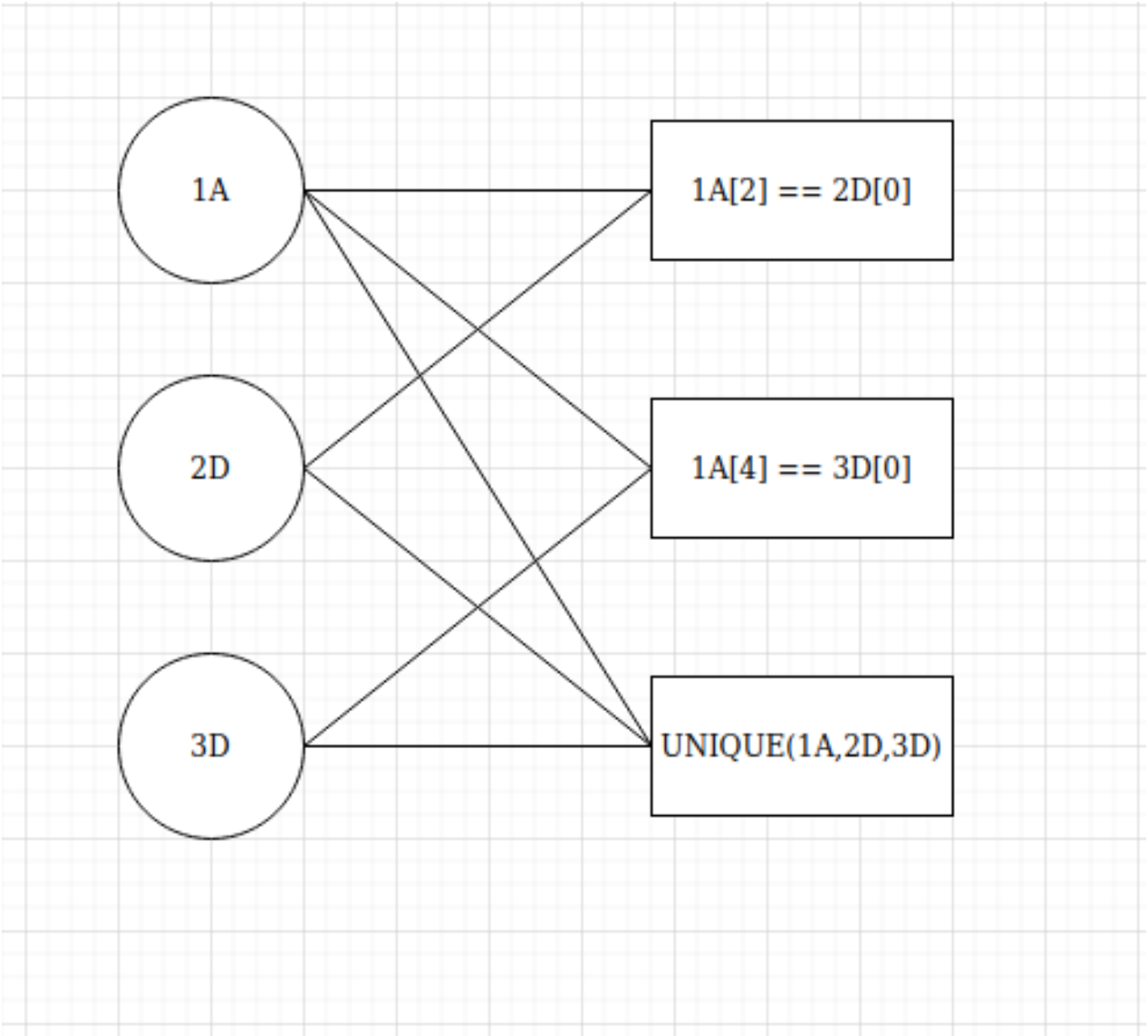
Exercise 4.2.

a) Start to sketch a constraint graph for the crossword CSP (you could use a tool like diagrams.net). Choose one variable, then sketch out the the constraints that variable occurs in, and finally include any other variables that are also in scope of the constraints you have already drawn. The full constraint graph will be too large to draw easily.

b) Apply *domain consistency* to this CSP, and restate any variable domains that change. Does the structure of the constraint graph change?

4.2 a

4.2 a



Domain consistency

Domain consistency

- Prune the domains as much as possible before selecting values from them.

Domain consistency

- Prune the domains as much as possible before selecting values from them.
- A variable is **Domain consistent** if no value if the domain of the node is ruled impossible by any on the constraints.

Domain consistency

- Prune the domains as much as possible before selecting values from them.
- A variable is **Domain consistent** if no value if the domain of the node is ruled impossible by any on the constraints.

Example: Is the scheduling example domain consistent?
Variables: $X = \{A, B, C, D, E\}$ that represent the starting times of various activities.
Domains: $dom_1 = \{1, 2, 3, 4\}, \forall i \in X$
Constraints: $(B \neq 3), (C \neq 2), (A \neq B), (B \neq C), (C < D)$
$(A = D), (E < A), (E < B), (E < C), (E < D), (B \neq D)$

No, $dom_B = \{1, 2, 3, 4\}$ isn't domain consistent as $B = 3$ violates the constraint $B \neq 3$.

4.2 b

Apply domain consistency to this CSP, and restate any variable domains that change. Does the structure of the constraint graph change?

4.2 b

Apply domain consistency to this CSP, and restate any variable domains that change. Does the structure of the constraint graph change?

- Domains are reduced to those with answers with word lengths that are consistent with the number of blank spaces, so all variable domains are changed.

4.2 b

Apply domain consistency to this CSP, and restate any variable domains that change. Does the structure of the constraint graph change?

- Domains are reduced to those with answers with word lengths that are consistent with the number of blank spaces, so all variable domains are changed.
- Updated domains:

4.2 b

Apply domain consistency to this CSP, and restate any variable domains that change. Does the structure of the constraint graph change?

- Domains are reduced to those with answers with word lengths that are consistent with the number of blank spaces, so all variable domains are changed.
- Updated domains:
  - $dom[1A] = \{HOSES, LASER, SAILS, SHEET, STEER\}$

4.2 b

Apply domain consistency to this CSP, and restate any variable domains that change. Does the structure of the constraint graph change?

- Domains are reduced to those with answers with word lengths that are consistent with the number of blank spaces, so all variable domains are changed.
- Updated domains:
  - $dom[1A] = \{HOSES, LASER, SAILS, SHEET, STEER\}$
  - $dom[2D] = \{HOSES, LASER, SAILS, SHEET, STEER\}$
  - $dom[3D] = \{HOSES, LASER, SAILS, SHEET, STEER\}$
  - $dom[4A] = \{HEEL, HIKE, KEEL, KNOT, LINE\}$
  - $dom[5D] = \{HEEL, HIKE, KEEL, KNOT, LINE\}$
  - $dom[6D] = \{AFT, ALE, EEL, LEE, TIE\}$
  - $dom[7A] = \{AFT, ALE, EEL, LEE, TIE\}$
  - $dom[8A] = \{HOSES, LASER, SAILS, SHEET, STEER\}$

4.2 b

Apply domain consistency to this CSP, and restate any variable domains that change. Does the structure of the constraint graph change?

- Domains are reduced to those with answers with word lengths that are consistent with the number of blank spaces, so all variable domains are changed.
- Updated domains:
  - $dom[1A] = \{HOSES, LASER, SAILS, SHEET, STEER\}$
  - $dom[2D] = \{HOSES, LASER, SAILS, SHEET, STEER\}$
  - $dom[3D] = \{HOSES, LASER, SAILS, SHEET, STEER\}$
  - $dom[4A] = \{HEEL, HIKE, KEEL, KNOT, LINE\}$
  - $dom[5D] = \{HEEL, HIKE, KEEL, KNOT, LINE\}$
  - $dom[6D] = \{AFT, ALE, EEL, LEE, TIE\}$
  - $dom[7A] = \{AFT, ALE, EEL, LEE, TIE\}$
  - $dom[8A] = \{HOSES, LASER, SAILS, SHEET, STEER\}$
- **Does the structure of the graph change?**

Backtracking search

Backtracking search

- Systematically explore domain by instantiating the variables one at a time.

Backtracking search

- Systematically explore domain by instantiating the variables one at a time.
- Evaluate each constraint predicate as soon as all its variables are bound.

Backtracking search

- Systematically explore domain by instantiating the variables one at a time.
- Evaluate each constraint predicate as soon as all its variables are bound.
- Any partial assignment that dosen't satisfy the constraint can be pruned.

Backtracking search

- Systematically explore domain by instantiating the variables one at a time.
- Evaluate each constraint predicate as soon as all its variables are bound.
- Any partial assignment that dosen't satisfy the constraint can be pruned.
- Essentially we "brute force" our way to a solution.

Sudoku backtracking example

4	2	6	5	7	1	3	9	8
8	5	7	2		3			6
	3			6			7	
		1				6		
5	4						1	9
		2				7		
	9			3			8	
2			8		4			7
	1		9		7		6	

(2,5)

(2,7)

(2,8)

⋮

Partial solution

4.3

Exercise 4.3. Apply *backtracking search* to the domain-consistent constraint graph (pseudocode given below). Record the number of nodes expanded in the search procedure. You can trace the algorithm manually, e.g. by sketching a search tree, or develop code to answer this question (reusing some of your tree search code from previous weeks).

```
function BACKTRACKING-SEARCH(csp) returns solution/failure
    return RECURSIVE-BACKTRACKING({ }, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
        if value is consistent with assignment given CONSTRAINTS[csp] then
            add {var = value} to assignment
            result ← RECURSIVE-BACKTRACKING(assignment, csp)
            if result ≠ failure then return result
            remove {var = value} from assignment
    return failure
```

4.3

4.3

- Select 1A, as the first unassigned variable.

4.3

- Select 1A, as the first unassigned variable.
- Check if the first value in domain of 1-across conflicts with the existing (empty) assignment; it dosen't, so expand 1A= *HOSES*.

4.3

- Select 1A, as the first unassigned variable.
- Check if the first value in domain of 1-across conflicts with the existing (empty) assignment; it dosen't, so expand 1A= *HOSES*.
- Now select 2-down, as the next unassigned variable; this is using the recursive function call.

4.3

- Select 1A, as the first unassigned variable.
- Check if the first value in domain of 1-across conflicts with the existing (empty) assignment; it dosen't, so expand 1A= *HOSES*.
- Now select 2-down, as the next unassigned variable; this is using the recursive function call.
- Check if the first value in domain of 2D, HOSES, conflicts with the existing assignment. It does! $\implies$ throw it away.

4.3

- Select 1A, as the first unassigned variable.
- Check if the first value in domain of 1-across conflicts with the existing (empty) assignment; it dosen't, so expand 1A= *HOSES*.
- Now select 2-down, as the next unassigned variable; this is using the recursive function call.
- Check if the first value in domain of 2D, HOSES, conflicts with the existing assignment. It does! $\implies$ throw it away.
- ... so on

4.3

- Select 1A, as the first unassigned variable.
- Check if the first value in domain of 1-across conflicts with the existing (empty) assignment; it dosen't, so expand 1A= *HOSES*.
- Now select 2-down, as the next unassigned variable; this is using the recursive function call.
- Check if the first value in domain of 2D, HOSES, conflicts with the existing assignment. It does! $\implies$ throw it away.
- ... so on

Arc-consistency

Definition (Arc consistency)

An arc $< X, r(X, \bar{Y}) >$ is arc consistent if, for each value $x \in dom(X)$, there is some value $\bar{y} \in Dom(\bar{Y})$ such that $r(x, \bar{y})$ is satisfied. A network is arc consistent if all arcs are arc consistent.

Arc-consistency

Definition (Arc consistency)

An arc $< X, r(X, \bar{Y}) >$ is arc consistent if, for each value $x \in dom(X)$, there is some value $\bar{y} \in Dom(\bar{Y})$ such that $r(x, \bar{y})$ is satisfied. A network is arc consistent if all arcs are arc consistent.

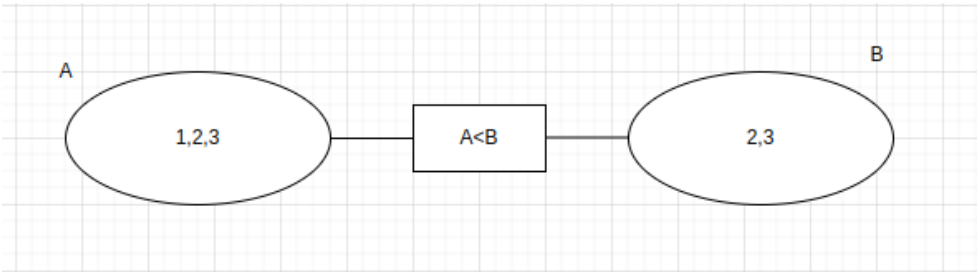
- Here $r(X, \bar{Y})$ is a constraint.

Arc-consistency

Definition (Arc consistency)

An arc $< X, r(X, \bar{Y}) >$ is arc consistent if, for each value $x \in dom(X)$, there is some value $\bar{y} \in Dom(\bar{Y})$ such that $r(x, \bar{y})$ is satisfied. A network is arc consistent if all arcs are arc consistent.

- Here $r(X, \bar{Y})$ is a constraint.



4.4

Exercise 4.4.

a) Apply *arc-consistency* to this CSP (manually or in code; pseudocode given below). Record the number of arc-consistency check operations that are performed. What is the outcome of applying arc consistency?

b) If needed, apply backtracking search to the arc-consistent CSP.

c) Compare the number of search expansion and/or consistency check operations of backtracking search and (arc-consistency + backtracking search).

One *efficient* algorithm for arc-consistency is commonly called AC-3 (Source: R&N textbook):

```
function AC-3(csp) returns false if an inconsistency is found and true otherwise
  inputs: csp, a binary CSP with components (X, D, C)
  local variables: queue, a queue of arcs, initially all the arcs in csp

  while queue is not empty do
    (X_i, X_j) ← REMOVE-FIRST(queue)
    if REVISE(csp, X_i, X_j) then
      if size of D_i = 0 then return false
      for each X_k in X_i.NEIGHBORS - {X_j} do
        add (X_k, X_i) to queue
  return true
```

```
function REVISE(csp, X_i, X_j) returns true iff we revise the domain of X_i
  revised ← false
  for each x in D_i do
    if no value y in D_j allows (x,y) to satisfy the constraint between X_i and X_j then
      delete x from D_i
      revised ← true
  return revised
```

Figure 6.3 The arc-consistency algorithm AC-3. After applying AC-3, either every arc is arc-consistent, or some variable has an empty domain, indicating that the CSP cannot be solved. The name "AC-3" was used by the algorithm's inventor (Mackworth, 1977) because it's the third version developed in the paper.

4.4

- 1A = {HOSES,LASER,SAILS,SHEET,STEER} pause
- Check for a value in the domain of 2D consistent with each element in 1A.
  2D = {HOSES,LASER,SAILS,SHEET,STEER}

4.4

- 1A = {HOSES,LASER,SAILS,SHEET,STEER} pause
- Check for a value in the domain of 2D consistent with each element in 1A.
  2D = {HOSES,LASER,SAILS,SHEET,STEER}
- Remove SAILS, SHEET, STEER from 1A.
  1A = {HOSES,LASER}

4.4

- 1A = {HOSES,LASER,SAILS,SHEET,STEER} pause
- Check for a value in the domain of 2D consistent with each element in 1A.
  2D = {HOSES,LASER,SAILS,SHEET,STEER}
- Remove SAILS, SHEET, STEER from 1A.
  1A = {HOSES,LASER}
- Check for a value in the domain of 3D consistent with each element in 1A.
  3D = {HOSES,LASER,SAILS,SHEET,STEER}

4.4

- 1A = {HOSES,LASER,SAILS,SHEET,STEER} pause
- Check for a value in the domain of 2D consistent with each element in 1A.
  2D = {HOSES,LASER,SAILS,SHEET,STEER}
- Remove SAILS, SHEET, STEER from 1A.
  1A = {HOSES,LASER}
- Check for a value in the domain of 3D consistent with each element in 1A.
  3D = {HOSES,LASER,SAILS,SHEET,STEER}
- Remove LASER from 1A
- This implies that 1A must be HOSES, as all other values are not arc consistent.
- Now move onto next variable...

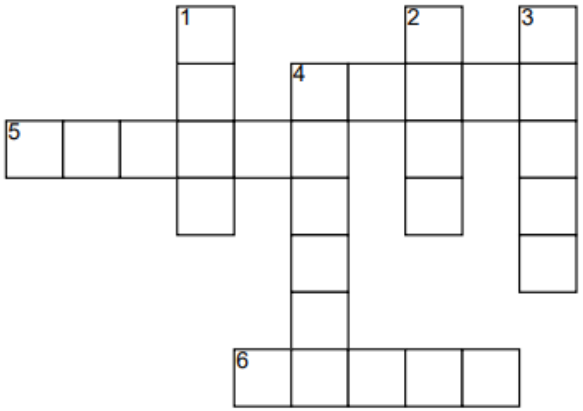
Code

https://gist.github.com/aryaman-sh/8f73ae3196c589f5b5bf6a1bf4c0a37e

2021 exam question

Question 3. **(20 marks)**

Consider the following crossword puzzle with candidate words, which can be cast as a constraint satisfaction problem (CSP):



Words:
ADA
NICK
MATT
ALINA
RENEE
DAVID
SERGE
ARCHIE
EMILIA
VEKTOR

Denote the variables: 1D (1-down), 2D, 3D, 4D, 4A (4-across), 5A and 6A. Note that words can only be used once in a solution.

- List all binary constraints.
- Apply domain consistency.
- Apply arc consistency.
- Apply backtracking to domain consistent CSP use ordering (1D,2D,3D,4D,4A,5A,6A)