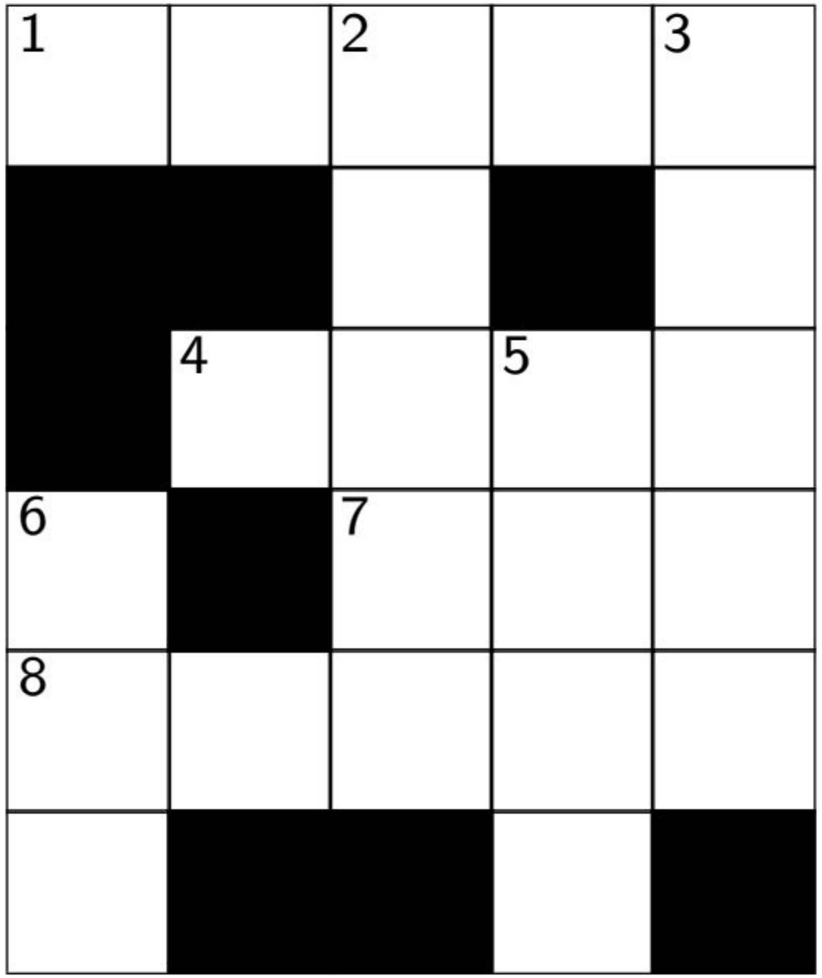
Motivation

- Constraint Satisfaction Problems (CSPs)
- View logic problems through a systematic framework (allows solutions to be applied to multiple problems and vice versa)
- Apply search algorithms to logic problems (Recursive Backtracking Search)
- Simplify logic problems by eliminating constraint violating solutions (Arc Consistency)

Crossword Puzzle Domain

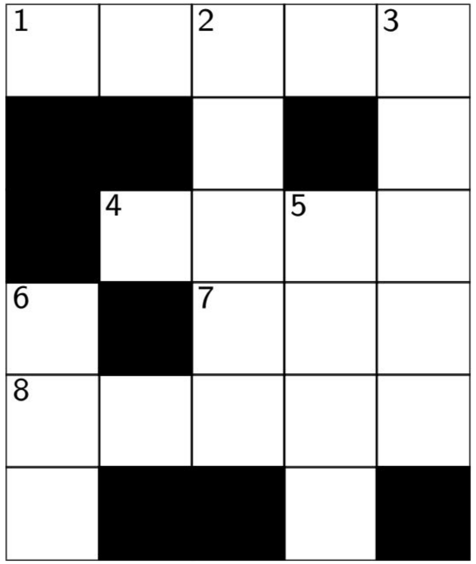


- Words: {AFT, ALE, EEL, HEEL, HIKE, HOSES, KEEL, KNOT, LASER, LEE, LINE, SAILS, SHEET, STEER, TIE}
- Words must start at a position label number and run left-to-right or top-to-bottom only
- Intersection boxes can contain only one letter
- Each word in the list provided can be used only once

Pause

CSP Representations

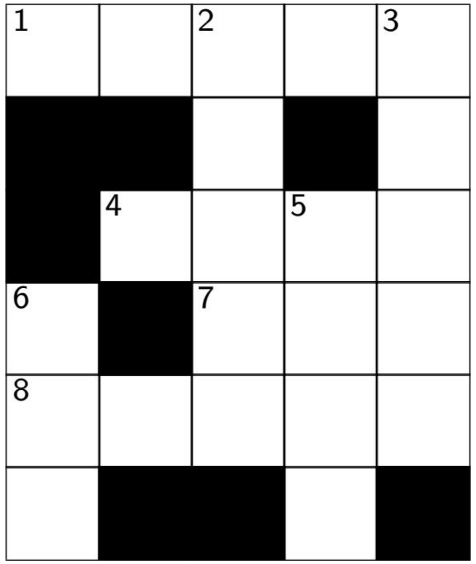
- What are the Variables and their Domains in this CSP?



{AFT, ALE, EEL, HEEL, HIKE, HOSES, KEEL, KNOT, LASER, LEE, LINE, SAILS, SHEET, STEER, TIE}

CSP Representations

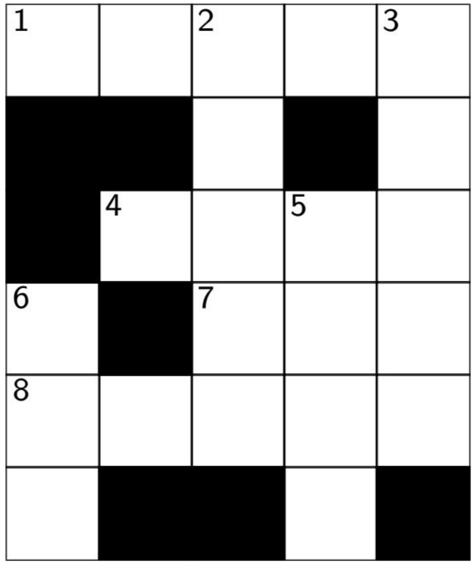
- What are the Variables and their Domains in this CSP?
  - **Variables:** 1-across, 2-down, 3-down, 4-across, 5-down, 6-down, 7-across and 8-across
  - **Domains:** {AFT, ALE, EEL, HEEL, HIKE, HOSES,KEEL, KNOT, LASER, LEE, LINE, SAILS, SHEET, STEER, TIE} for all variables



{AFT, ALE, EEL, HEEL, HIKE, HOSES, KEEL, KNOT, LASER, LEE, LINE, SAILS, SHEET, STEER, TIE}

CSP Representations

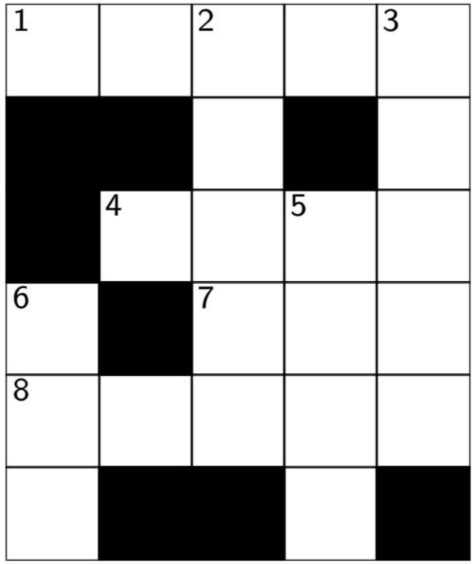
- What are the Constraints in this CSP?



{AFT, ALE, EEL, HEEL, HIKE, HOSES, KEEL, KNOT, LASER, LEE, LINE, SAILS, SHEET, STEER, TIE}

CSP Representations

- What are the Constraints in this CSP?
  - Variables can be set to only words of the correct length (these are domain constraints / 1-constraints)
  - Letters used at the intersection of two words must be equal (binary constraints / 2-constraints)
  - Each word is used only once
    - Is this a “global” constraint? Or is it a set of binary constraints, in which each pair of words are linked by a “not-equal-to” predicate?
    - It can be either – we usually prefer to use constraints with the fewest number of variables, but this problem is small enough that one global constraint is fine

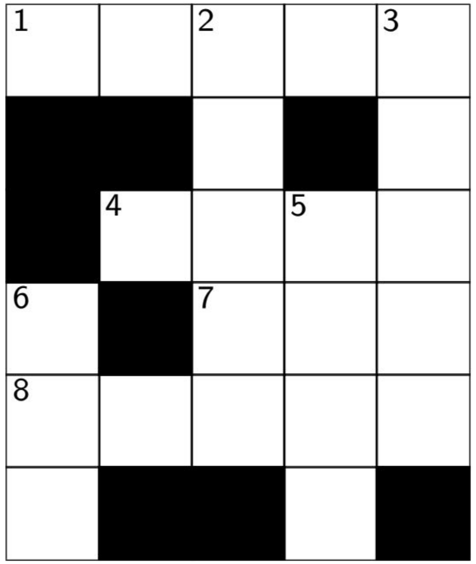


{AFT, ALE, EEL, HEEL, HIKE, HOSES, KEEL, KNOT, LASER, LEE, LINE, SAILS, SHEET, STEER, TIE}

Pause

CSP Representations

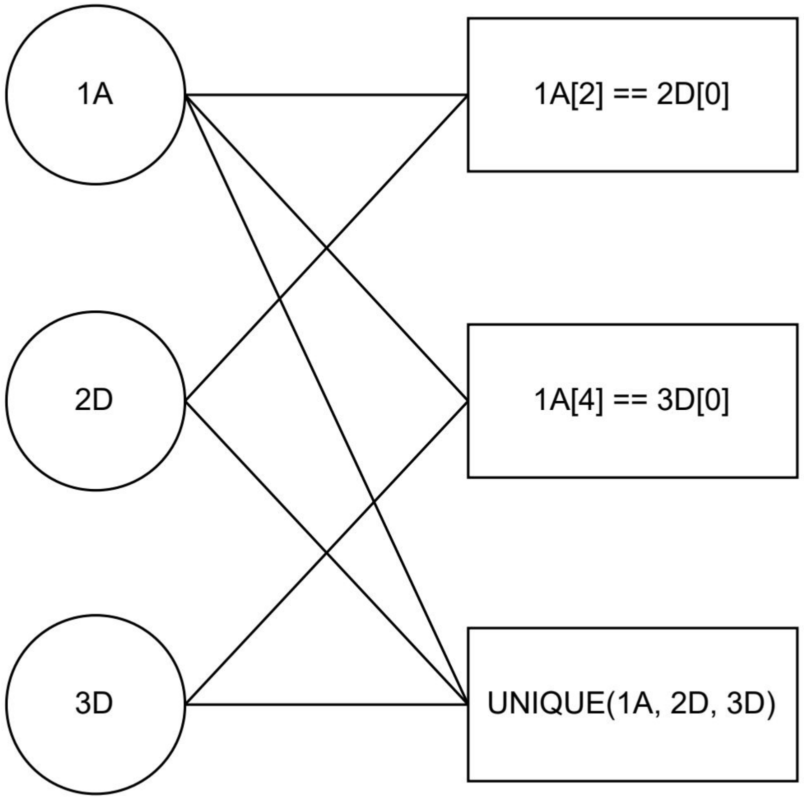
- Start to sketch a constraint graph for the crossword CSP
- Choose one variable, then sketch out the constraints that variable occurs in, and finally include any other variables that are also in scope of the constraints you have already drawn.
- The full constraint graph will be too large to draw easily



{AFT, ALE, EEL, HEEL, HIKE, HOSES, KEEL, KNOT, LASER, LEE, LINE, SAILS, SHEET, STEER, TIE}

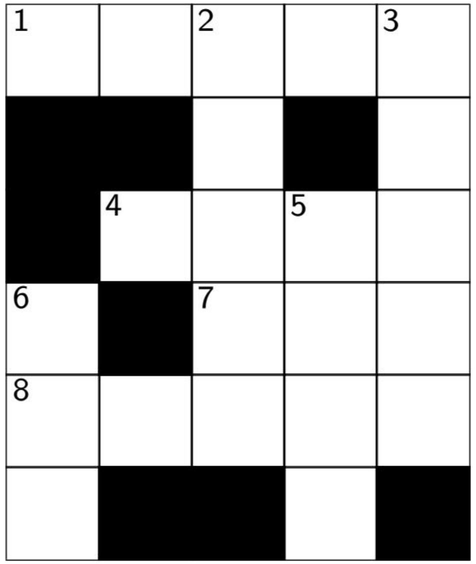
CSP Representations

- Start to sketch a constraint graph for the crossword CSP
- Use a bipartite graph
- A partial solution is shown, starting from 1A
- The coordinates refer to the grid in row-column from the top left corner (0-indexed)



CSP Representations

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



{AFT, ALE, EEL, HEEL, HIKE, HOSES, KEEL, KNOT, LASER, LEE, LINE, SAILS, SHEET, STEER, TIE}

CSP Representations

- Apply domain consistency to this CSP, and restate any variable domains that 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
  - Domains after applying node consistency:

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}

CSP Representations

- Does the structure of the constraint graph change?
  - No, as domain constraints (which apply to only 1 variable) are not represented in the constraint graph

Backtracking Search

- Apply backtracking search to the domain-consistent constraint graph
- 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
```

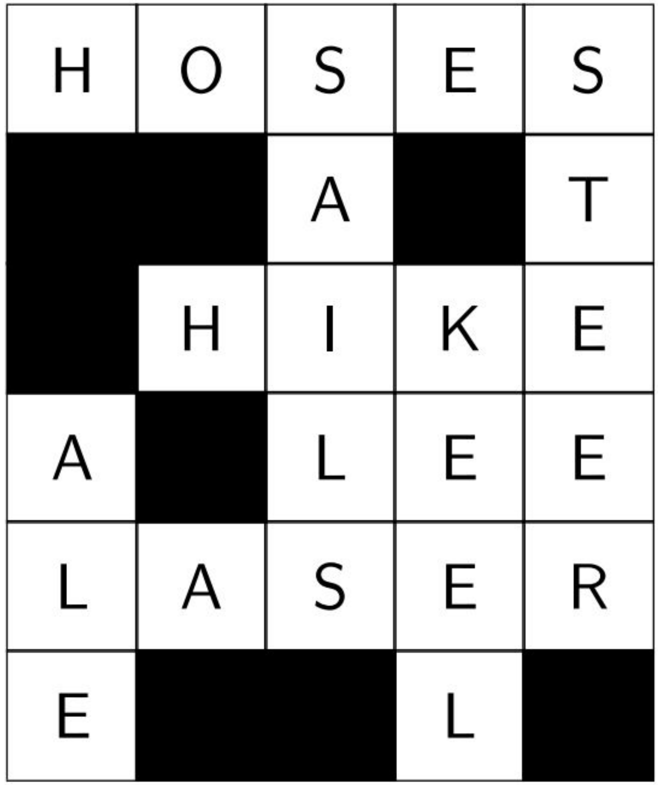
Backtracking Search

- As a demonstration, we can manually trace Backtracking-Search:

1. Select 1-across, as the first unassigned variable.
2. Check if the first value in the domain of 1-across conflicts with the existing (empty) assignment; it doesn't, so expand a node for 1-across=HOSES.
3. Now select 2-down, as the next unassigned variable; this is using the recursive function call.
4. Check if the first value in the domain of 2-down, HOSES, conflicts with the existing assignment — it does! Throw it away.
5. The second value, LASER also conflicts with the existing assignment, so throw it away. At this point we have iterated over the inner if statement twice.
6. The third value in the domain, 2-down=SAILS is consistent with 1-across=HOSES, so expand a child node of 1-across=HOSES given by 2-down=SAILS
7. Select 3-down as the next unassigned variable.
8. . . . and so on.

Backtracking Search

- Eventually, you will find the solution as:



Arc Consistency

- Apply arc-consistency to this CSP (manually or in code)
- Record the number of arc-consistency check operations that are performed
- What is the outcome of applying arc consistency?

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**
    $(X_i, X_j) \leftarrow$ 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 $\leftarrow$ *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 $\leftarrow$ *true*
  return *revised*

Arc Consistency

- The first few operations of AC-3:
  a) 1-across: {HOSES, LASER, SAILS, SHEET, STEER}
  b) Check for a value in dom(2-down) consistent with each element of dom(1-across) 2-down: {HOSES, LASER, SAILS, SHEET, STEER}
  c) Remove SAILS, SHEET, STEER from dom(1-across), because there is no value in dom(2-down) that is consistent with the value assignments.
     Set 1-across = {HOSES, LASER}
  d) Next, check for a value in dom(3-down) consistent with each element of dom(1-across) 3-down: {HOSES, LASER, SAILS, SHEET, STEER}
  e) Remove LASER from dom(1-across)
  f) 1-across must be HOSES, as all other values are not arc consistent.
  g) Now move on to the next variable (e.g. 2-down). . .

Arc Consistency

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

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

Arc Consistency

- If needed, apply backtracking search to the arc-consistent CSP
  - If you apply the arc-consistency algorithm correctly, you will find that it solves the cross-word puzzle, and no further search is required
  - Note that applying arc consistency will not always lead to a unique solution, so search on the reduced may still be required for some problems
- Compare the number of search expansion and/or consistency check operations of backtracking search and (arc-consistency + backtracking search)

Arc Consistency

- If needed, apply backtracking search to the arc-consistent CSP
  - If you apply the arc-consistency algorithm correctly, you will find that it solves the cross-word puzzle, and no further search is required
  - Note that applying arc consistency will not always lead to a unique solution, so search on the reduced may still be required for some problems
- Compare the number of search expansion and/or consistency check operations of backtracking search and (arc-consistency + backtracking search)
  - Even allowing for different variations on counting the number of operations, AC-3 solves the problem in many fewer operations than backtracking search