**CS 6613** **Fall 2023**
**HW 3** **E. K. Wong**

*Total # questions = 4. Total # points = 60. Figures 1 to 3 are on pages 2 and 3.*

**1. [10 points]** (a) What are the *time complexity* and *space complexity* if the regular Depth-First Search algorithm in Chapter 3 is used to solve constrain satisfaction problems? (b) What are the *time complexity* and *space complexity* if the Backtracking-Search Algorithm in Figure 2 below is used to solve constrain satisfaction problems? Define the parameters you used for both parts (a) and (b).

**2. [15 points]** In the map coloring problem in Figure 1 below, suppose the people in region NT do not like the *red* and *green* colors. The domain for NT therefore contains only the *blue* color, and the domains for all other regions contain the colors *red*, *green* and *blue*. Apply the Backtracking-Search algorithm in Figure 2 below to solve this problem and draw the search tree produced. When applying the algorithm, use the *minimum remaining value* and *degree heuristics* for selecting variables in the SELECT-UNASSIGNED-VARIABLE function. For the ORDER-DOMAIN-VALUES function, simply order the values in the order *red*, *green* and *blue* instead of using heuristics. You can skip the INFERENCE function when applying the Backtracking-Search algorithm.

**3. [15 points]** Given the specifications of the cryptarithmetic problem below, where x1 and x2 are auxiliary variables representing the carry overs. Assuming the Backtracking-Search algorithm in Figure 2 is used to solve this problem, use the *minimum remaining values* and *degree heuristics* to select the variable to work on at each level of the search tree. You can skip the INFERENCE function in the Backtracking-Search algorithm. Let the root node be at level 0. List the variables selected at levels 1 to *n* of the search tree, where *n* is the number of variables in this problem. Show all work to get full credits. (You do not have to draw a search tree and find the solution to this problem. Just determine the variable selected at each level using the two heuristics.)

Variables:

```
   x2x1
    ONE
 +  ONE
 ----------
    TWO
```

Domains:

- O: {2,4}
- T: {2,3,4,5,6,7,8,9}
- N, E and W: {0,1,2,3,4,5,6,7,8,9}
- x1 and x2: {0,1}

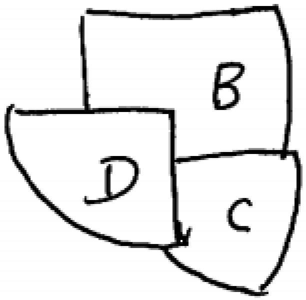
Constraints:

*Alldiff*(O,N,E,T,W)
E + E = 10*x1 + O
x1 + N + N = 10*x2 + W
x2 + O + O = T

**4. [20 points]** Given the map coloring problem below, the goal is to color regions *B*, *C* and *D* so that no two adjacent regions have the same color. The initial domain values for the three regions are given in the table below.

(a) Region *C* has only the color *R* in its domain. We can therefore assign the color *R* to region *C*. Apply Forward Checking (one time) to update the domain values of region *C's* neighbors.
(b) Apply the *AC-3* algorithm in Figure 3 to the three regions with initial domain values given in the top row of the table below. List the initial content of the *queue* after it is initialized with all the arcs in the CSP. List the domain values of each region after the algorithm stops.

(Note: (a) and (b) above are independent sub-problems. You are to start with the initial domain values in the top row of the table below for each sub-problem.)




| | B | C | D |
|---|---|---|---|
| Initial domain values | RG | R | RGB |
| After Forward Checking | | | |
| After AC-3 | | | |

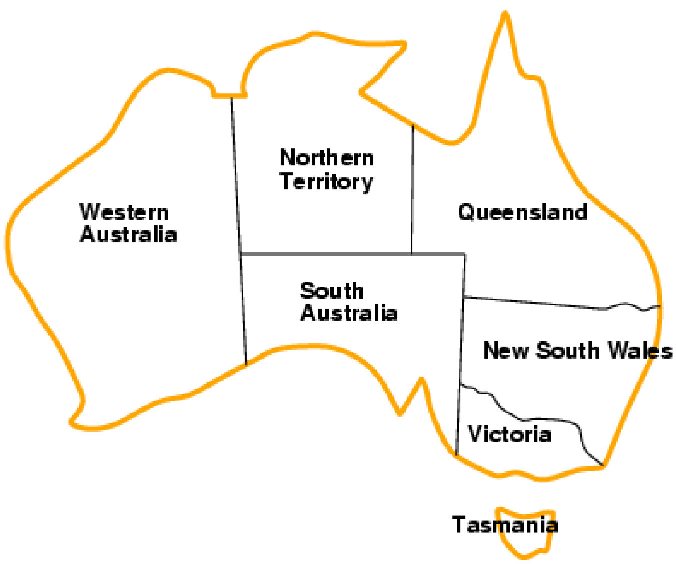



**Figure 1. Map of Australia**

```
function BACKTRACKING-SEARCH(csp) returns a solution or failure
  return BACKTRACK(csp, { })

function BACKTRACK(csp, assignment) returns a solution or failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(csp, assignment)
  for each value in ORDER-DOMAIN-VALUES(csp, var, assignment) do
     if value is consistent with assignment then
        add {var = value} to assignment
        inferences ← INFERENCE(csp, var, assignment)
        if inferences ≠ failure then
           add inferences to csp
           result ← BACKTRACK(csp, assignment)
           if result ≠ failure then return result
           remove inferences from csp
        remove {var = value} from assignment
  return failure
```

**Figure 2. The Backtracking-Search Algorithm for CSPs**

```
function AC-3(csp) returns false if an inconsistency is found and true otherwise
  queue ← a queue of arcs, initially all the arcs in csp

  while queue is not empty do
    (X_i, X_j) ← POP(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 3. The AC-3 algorithm.**