Midterm Review

Summer 2019

General Information

All information is on CSC384 web page **Test** tab at the top of the page.

- 50 minutes in duration
- Starts at 6:10 and ends at 7:00
- no aids permitted
- worth 16% of your course grade

Tips and Resources

- Let the **lecture slides and the posted tutorial materials** be your guide for studying. If you're unclear about something, augment it with the text or other online materials.
- Make sure you *understand* the material. If there are proofs, work through them so you understand them. Understand the rationale for why things work the way they do.
- Work through some problem sets. Look at the **posted sample problems on the test web page** as well as problems we went through in class.
- Know and understand the facts: know the complexity of different algorithms and why.

Topics

1. Uninformed and Informed Search
2. Game Tree Search
3. Backtracking and CSPs

Note that Local Search will not be on the exam.

Also, no problem solving activities related to MCTS (only short answer).

Uninformed and Informed Search

- Breadth-first search
- Depth-first search
- Depth-limited search
- Uniform Cost Search
- A* search
- IDA* search

- Cycle Checking, Path Checking

Uninformed and Informed Search

Metrics:

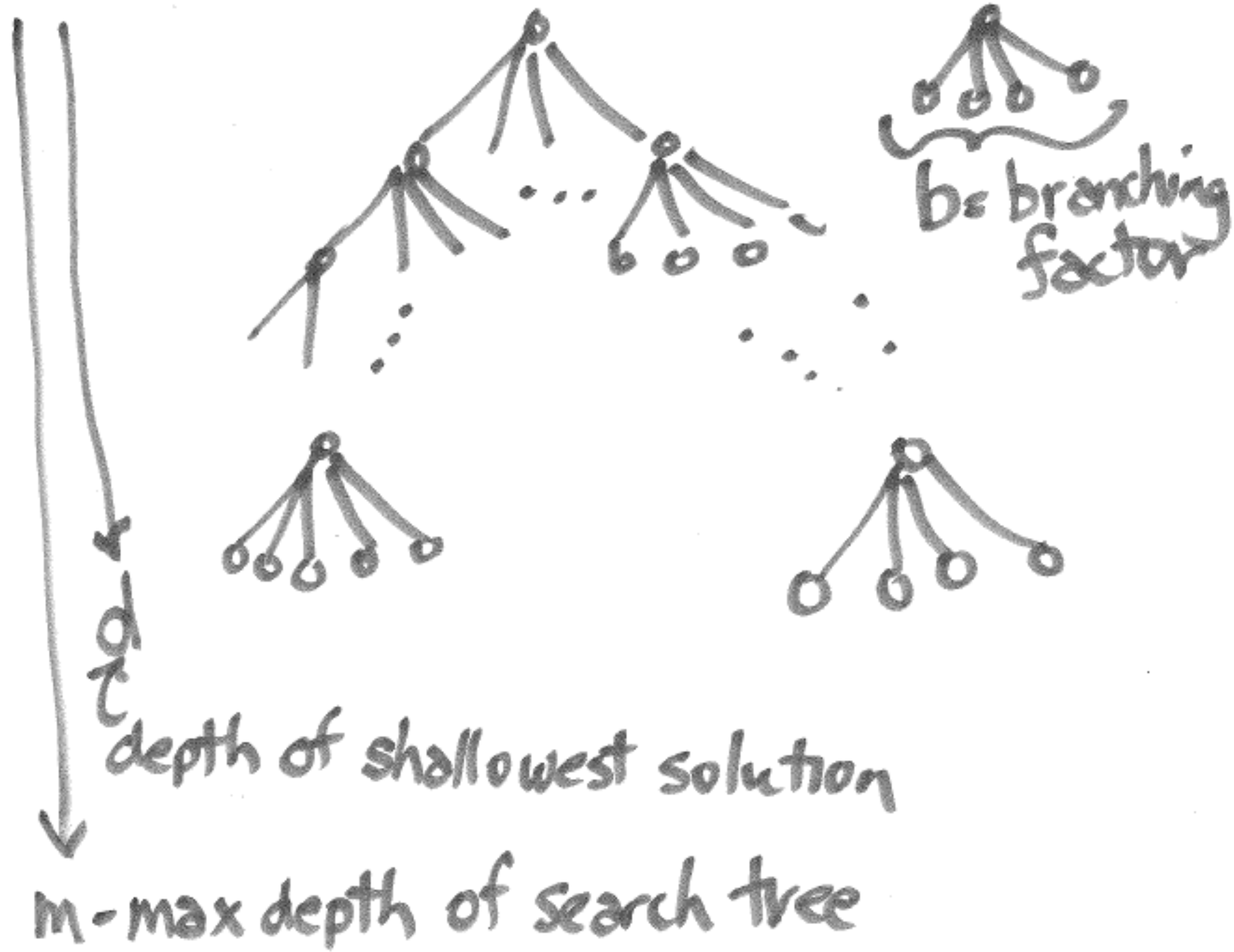
- Completeness
- Space Complexity
- Time Complexity
- Optimality

Search: The Basic Algorithm

```
TreeSearch(Frontier, Sucessors, Goal? )
    If Frontier is empty return failure
    Curr = select state from Frontier
    If (Goal?(Curr)) return Curr.
    Frontier' = (Frontier - {Curr}) U Successors(Curr)
    return TreeSearch(Frontier', Successors, Goal?)
```

REMEMBER: The different forms of search just boil down to the order you put nodes on the frontier/open list.

The Parameters of Search



From Russell and Norvig

3.4.7 Comparing uninformed search strategies

Figure 3.21 compares search strategies in terms of the four evaluation criteria set forth in Section 3.3.2. This comparison is for tree-search versions. For graph searches, the main differences are that depth-first search is complete for finite state spaces and that the space and time complexities are bounded by the size of the state space.

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes^{a}	Yes^{a,b}	No	No	Yes^{a}	Yes^{a,d}
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes^{c}	Yes	No	No	Yes^{c}	Yes^{c,d}

Figure 3.21 Evaluation of tree-search strategies. b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are as follows: ^{a} complete if b is finite; ^{b} complete if step costs $\geq \epsilon$ for positive ϵ; ^{c} optimal if step costs are all identical; ^{d} if both directions use breadth-first search.

This is slightly different from what we determined in class/notes (e.g., $O(b^d)$ vs $O(b^{d+1})$) Why? Because they allow for a goal check *before* putting a node on the frontier/open list, whereas we do it when we pop the node off. Think about the implications of moving the goal check with respect to optimality.

Heuristic Search Techniques

- Greedy Best First Search, A* Search, IDA*
- Time Complexity?
- Space Complexity?
- Optimal?
- Complete?

- Admissible Heuristics, Monotonic (Consistent) Heuristics

- Optimality & how it interacts with cycle checking

Topics

1. Uninformed and Informed Search
2. **Game Tree Search**
3. Backtracking and CSPs

Game Tree Search

- Definitions: Two-players, Discrete, Finite, Zero-Sum, Deterministic, Perfect Information
- Components of Two-Player Zero-Sum Game: players, states, terminals, successors, utilities
- Minimax Strategy
- DFS Implementation (Time and Space complexity?)
- Alpha-beta pruning (Empirical savings?)
- Heuristics for games.

Game Tree Search

Depth-First Implementation of MiniMax

```
DFMiniMax(n, Player) //return Utility of state n given that
      //Player is MIN or MAX

 If n is TERMINAL
 Return V(n) //Return terminal states utility
         //(V is specified as part of game)

 //Apply Player's moves to get successor states.
 ChildList = n.Successors(Player)
 If Player == MIN
  return minimum of DFMiniMax(c, MAX) over c  ∈ChildList
 Else //Player is MAX
  return maximum of DFMiniMax(c, MIN) over c  ∈ChildList
```

Game Tree Search

Implementing Alpha-Beta Pruning

```
AlphaBeta(n,Player,alpha,beta) //return Utility of state
If n is TERMINAL
  return V(n) //Return terminal states utility
ChildList = n.Successors(Player)
If Player == MAX
 for c in ChildList
  alpha = max(alpha, AlphaBeta(c,MIN,alpha,beta))
  If beta <= alpha
    break
 return alpha
Else //Player == MIN
  for c in ChildList
  beta = min(beta, AlphaBeta(c,MAX,alpha,beta))
  If beta <= alpha
    break
 return beta
```

Topics

1. Uninformed and Informed Search
2. Game Tree Search
3. **Backtracking and CSPs**

Constraint Satisfaction Problems

- Contrasting CSP and a Search
- Vectors of features as states, each with a domain
- Definitions: Unary, Binary or n-ary Constraints, Scope, Partial State, Domain Wipe Out, Support (for a variable assignment)
- Problem formulations (e.g., binary vs n-ary constraints)and implications of different
- Search augmented with inference (constraint propagation) and trade offs

Backtracking Search

```
BT(Level)
  If all variables assigned
       PRINT Value of each Variable
       RETURN or EXIT (RETURN for more solutions)
                      (EXIT for only one solution)
  V := PickUnassignedVariable()
  Assigned[V] := TRUE
  for d := each member of Domain(V)  (the domain values of V)
       Value[V] := d
       ConstraintsOK = TRUE
       for each constraint C such that
               a) V is a variable of C and
               b) all other variables of C are assigned:
           IF C is not satisfied by the set of current
               assignments:
               ConstraintsOK = FALSE
       If ConstraintsOk == TRUE:
           BT(Level+1)

  Assigned[V] := FALSE //UNDO as we have tried all of V's values
  return
```

Constraint Propagation (Inference)

- Forward Checking and GAC
- Variable Ordering Heuristics (MRV)
- Arc Consistency (of a CSP, etc.)

Backtracking Search: The Algorithm BT

```
BT(Level)
  If all variables assigned
      PRINT Value of each Variable
      RETURN or EXIT (RETURN for more solutions)
                     (EXIT for only one solution)
  V := PickUnassignedVariable()
  Assigned[V] := TRUE
  for d := each member of Domain(V)  (the domain values of V)
      Value[V] := d
      ConstraintsOK = TRUE
      for each constraint C such that
              a) V is a variable of C and
              b) all other variables of C are assigned:
                ;(rarely the case initially high in the search tree)
           IF C is not satisfied by the set of current
              assignments:
              ConstraintsOK = FALSE
      If ConstraintsOk == TRUE:
           BT(Level+1)

  Assigned[V] := FALSE //UNDO as we have tried all of V's values
  return
```

Forward Checking Algorithm

```
FC(Level) /*Forward Checking Algorithm */
   If all variables are assigned
        PRINT Value of each Variable
        RETURN or EXIT (RETURN for more solutions)
                       (EXIT for only one solution)
   V := PickAnUnassignedVariable()
   Assigned[V] := TRUE
   for d := each member of CurDom(V)
        Value[V] := d
        DWOoccured:= False
        for each constraint C over V such that
               a) C has only one unassigned variable X in its scope
           if(FCCheck(C,X) == DWO)    /* X domain becomes empty*/
                DWOoccurred:= True
                break /* stop checking constraints */
        if(not DWOoccured) /*all constraints were ok*/
           FC(Level+1)
        RestoreAllValuesPrunedByFCCheck()
   Assigned[V] := FALSE //undo since we have tried all of V's values
   return;
```

Forward Checking Algorithm

For a single constraint C:

```
FCCheck(C,x)
  // C is a constraint with all its variables already
  // assigned, except for variable x.
  for d := each member of CurDom[x]
     IF making x = d together with previous assignments
        to variables in scope C falsifies C
     THEN remove d from CurDom[x]
 IF CurDom[x] = {} then return DWO (Domain Wipe Out)
 ELSE return ok
```

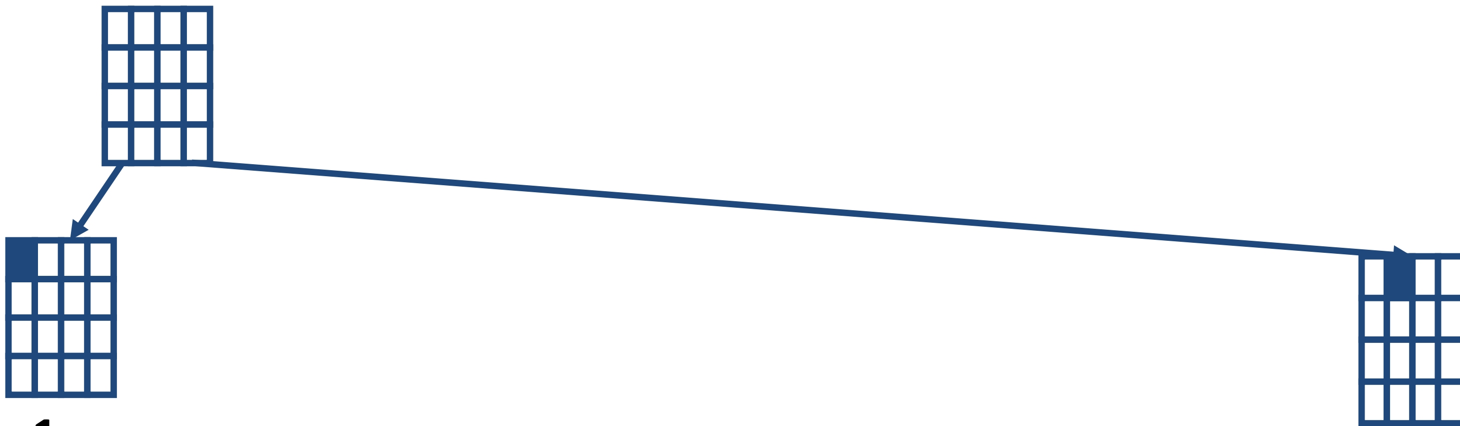
GAC Algorithm, enforce GAC during search

```
GAC(Level) /*Maintain GAC Algorithm */
   If all variables are assigned
      PRINT Value of each Variable
      RETURN or EXIT (RETURN for more solutions)
                     (EXIT for only one solution)
   V := PickAnUnassignedVariable()
   Assigned[V] := TRUE
   for d := each member of CurDom(V)
      Value[V] := d
      Prune all values of V ≠ d from CurDom[V]
      for each constraint C whose scope contains V
        Put C on GACQueue
      if(GAC_Enforce() != DWO)
        GAC(Level+1) /*all constraints were ok*/
      RestoreAllValuesPrunedFromCurDoms()
   Assigned[V] := FALSE
   return;
```

Enforce GAC (prune all GAC inconsistent values)

```
GAC_Enforce()
   // GAC-Queue contains all constraints one of whose variables has
   // had its domain reduced. At the root of the search tree
   // first we run GAC_Enforce with all constraints on GAC-Queue

  while GACQueue not empty

      C = GACQueue.extract()

      for V := each member of scope(C)
          for d := CurDom[V]

               Find an assignment A for all other
               variables in scope(C) such that
               C(A ∪ V=d) = True
               if A not found

                  CurDom[V] = CurDom[V] – d
                  if CurDom[V] = ∅
                       empty GACQueue
                       return DWO //return immediately

                  else

                       push all constraints C′ such that
                       V  ∈ scope(C′) and C′ ∉ GACQueue
                       on to GACQueue
   return TRUE //while loop exited without DWO
```

Example: N-Queens GAC search Space



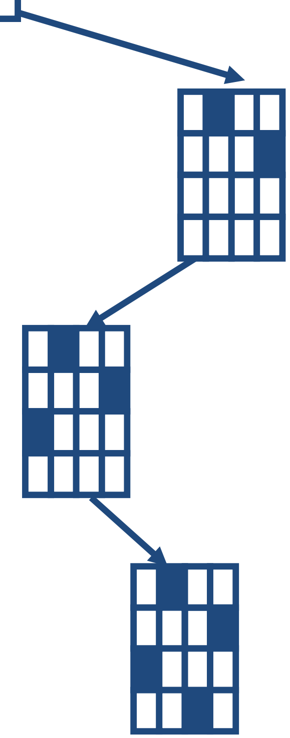
V1 = 1

arc consistency stages:

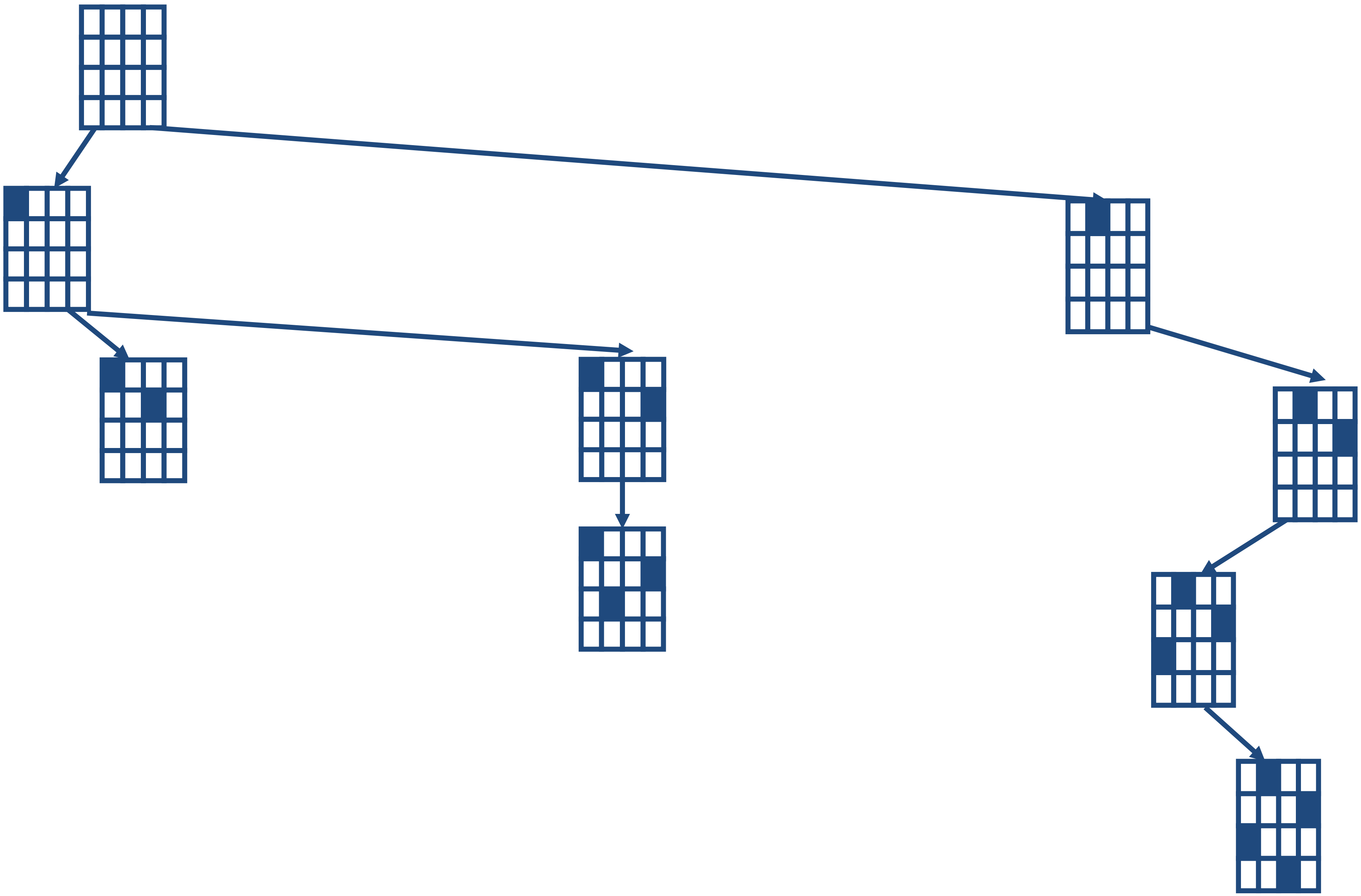
1. V2 = {3,4}, V3 = {2,4}, V4 = {2, 3}
   V2=1,2 & V3 = 1,3 & V3 = 1,4 are inconsistent with V1=1.
2. V2 = {4} (**V2=3 is inconsistent** with both values in CurDom[V3]
3. V3 = {2} (**V3 = 2 is inconsistent with values in CurDom[V2]**
4. V4 = {} (both values for V4 inconsistent with values in CurDom[V3]

DWO

Recall we kept Exploring with FC



CONTRAST TO: N-Queens FC search Space



CONTRAST : N-Queens Backtracking Search Space