EECS 3401: Intro to AI & LP
CSPs & Backtracking Search

- Readings: R & N Chapter 6.

Constraint Satisfaction Problems

- The search algorithms we discussed so far had no knowledge of the states representation (black box). So, we could not take advantage of domain-specific information.
- CSP are a special class of search problems with a uniform and simple state representation.
- This allows to design more efficient algorithms.

Constraint Satisfaction Problems

- Many problems can be represented as a search for a vector of feature values.
  - k-features: variables.
  - Each feature has a value. Domain of values for the variables.
  - e.g., height = {short, average, tall}, weight = {light, average, heavy}.
- In these problems the problem is to search for a set of values for the features (variables) so that the values satisfy some conditions (constraints).

Constraint Satisfaction Problems

- Sudoku:
  - 81 variables, the value in each cell.
  - Values: a fixed value for those cells that are already filled in, the values {1-9} for those cells that are empty.
  - Solution: a value for each cell satisfying the constraints:
    - no cell in the same column can have the same value.
    - no cell in the same row can have the same value.
    - no cell in the same sub-square can have the same value.

Constraint Satisfaction Problems

- Scheduling
  - Want to schedule a time and a space for each final exam so that
    - No student is scheduled to take more than one final at the same time.
    - The space allocated has to be available at the time set.
    - The space has to be large enough to accommodate all of the students taking the exam.

Constraint Satisfaction Problems

- Variables:
  - T1, …, Tm: Ti is a variable representing the scheduled time for the i-th final.
    - Assume domains are fixed to {MonAm, MonPm, …, FriAm, FriPm}.
  - S1, …, Sm: Si is the space variable for the i-th final.
    - Domain of Si are all rooms big enough to hold the i-th final.

Constraint Satisfaction Problems

- Want to find an assignment of values to each variable (times, rooms for each final), subject to the constraints:
  - For all pairs of finals i, j such that there is a student taking both:
    - Ti ≠ Tj
  - For all pairs of finals i, j
    - Ti ≠ Tj or Si ≠ Sj
      - either i and j are not scheduled at the same time, or if they are they are not in the same space.

Constraint Satisfaction Problems (CSP)

- More formally.
- A CSP consists of
  - a set of variables V1, …, Vn
  - for each variable a domain of possible values Dom[Vi].
  - A set of constraints C1,…, Cm.

Constraint Satisfaction Problems

- Each variable be assigned any value from its domain.
  - Vi = d where d ∈ Dom[Vi]
- Each constraint C has
  - A set of variables it is over, called its scope: e.g., C(V1,V2,V4).
  - Is a boolean function that maps assignments to these variables to true/false.
    - e.g. C(V1=a,V2=b,V4=c) = True
      - this set of assignments satisfies the constraint.
    - e.g. C(V1=b,V2=c,V4=c) = False
      - this set of assignments falsifies the constraint.

Constraint Satisfaction Problems

- A solution to a CSP is
  - an assignment of a value to all of the variables such that
    - every constraint is satisfied.

Constraint Satisfaction Problems

- Sudoku:
  - $V_{11}, V_{12}, \ldots, V_{21}, V_{22}, \ldots, V_{91}, \ldots, V_{99}$
    - $Dom[V_{ij}] = \{1-9\}$ for empty cells
    - $Dom[V_{ij}] = \{k\}$ a fixed value k for filled cells.
  - Row constraints:
    - $CR1(V_{11}, V_{12}, V_{13}, \ldots, V_{19})$
    - $CR2(V_{21}, V_{22}, V_{23}, \ldots, V_{29})$
    - ...., $CR9(V_{91}, V_{92}, \ldots, V_{99})$
  - Column Constraints:
    - $CC1(V_{11}, V_{21}, V_{31}, \ldots, V_{91})$
    - $CC2(V_{21}, V_{22}, V_{13}, \ldots, V_{92})$
    - ...., $CC9(V_{19}, V_{29}, \ldots, V_{99})$
  - Sub-Square Constraints:
    - $CSS1(V_{11}, V_{12}, V_{13}, V_{21}, V_{22}, V_{23}, V_{31}, V_{32}, V_{33})$
    - $CSS1(V_{14}, V_{15}, V_{16}, \ldots, V_{34}, V_{35}, V_{36})$

Constraint Satisfaction Problems

- Sudoku:
  - Each of these constraints is over 9 variables, and they are all the same constraint:
    - Any assignment to these 9 variables such that each variable has a unique value satisfies the constraint.
    - Any assignment where two or more variables have the same value falsifies the constraint.
  - Such constraints are often called ALL-DIFF constraints.

Constraint Satisfaction Problems

- Sudoku:
  - Thus Sudoku has 3x9 ALL-Diff constraints, one over each set of variables in the same row, one over each set of variables in the same column, and one over each set of variables in the same sub-square.
  - Note also that an ALL-Diff constraint over k variables can be equivalently represented by k choose 2 not-equal constraints over each pair of these variables.
    - e.g. CSS1(V_{11}, V_{12}, V_{13}, V_{21}, V_{22}, V_{23}, V_{31}, V_{32}, V_{33}) = NEQ(V_{11},V_{12}), NEQ(V_{11},V_{13}), NEQ(V_{11},V_{21}) …, NEQ(V_{32},V_{33})
      - NEQ is a not-equal constraint.

Constraint Satisfaction Problems

- Exam Scheduling
  - constraints:
  - For all pairs of finals i, j such that there is a student taking both:
    - NEQ(Ti,Tj)
  - For all pairs of finals i, j
    - C(Ti,Tj,Si,Sj)
      - This constraint is satisfied
        - by any set of assignments in which Ti $\neq$ Tj.
        - any set of assignments in which Si $\neq$ Sj.
      - Falsified by any set of assignments in which Ti=Tj as well as Si=Sj.

Solving CSPs

- CSPs can be solved by a specialized version of depth first search.
- Key intuitions:
  - We can build up to a solution by searching through the space of partial assignments.
  - Order in which we assign the variables does not matter---eventually they all have to be assigned.
  - If during the process of building up a solution we falsify a constraint, we can immediately reject all possible ways of extending the current partial assignment.

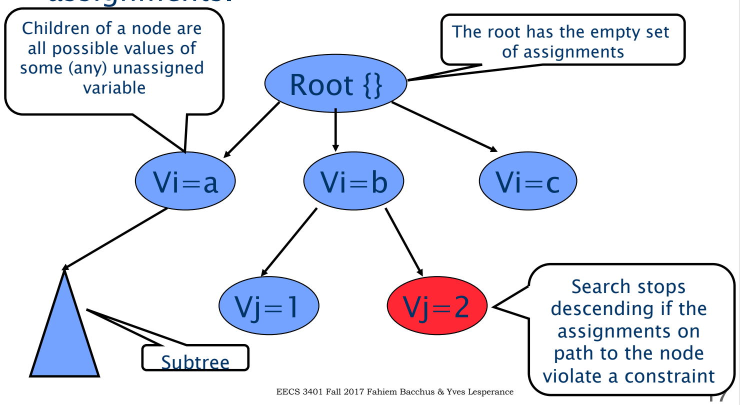
Backtracking Search

- These ideas lead to the backtracking search algorithm

```
Algorithm BT (Backtracking)
  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()
    Variable[Level] := V
    Assigned[V] := TRUE
    for d := each member of Domain(V)
       Value[V] := d
       OK := TRUE
       for each constraint C such that
                V is a variable of C
                and all other variables of C are assigned.
          if C is not satisfied by the current set of assignments
              OK := FALSE
       if(OK)
           BT(Level+1)
    return
```

Solving CSPs

- The algorithm searches a tree of partial assignments.



Backtracking Search

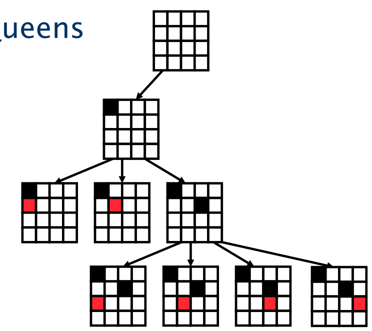
- Heuristics are used to determine which variable to assign next "`PickUnassignedVariable`".
- The choice can vary from branch to branch, e.g.,
  - under the assignment V1=a we might choose to assign V4 next, while under V1=b we might choose to assign V5 next.
- This "dynamically" chosen variable ordering has a tremendous impact on performance.

Example.

- N-Queens. Place N Queens on an N X N chess board so that no Queen can attack any other Queen.
  - Variables, one per row.
    - Value of Vi is the column the Queen in row i is place.
  - Constrants.
    - Vi ≠ Vj for all i ≠ j (can put two Queens in same column)
    - |Vi-Vj| ≠ i-j (Diagonal constraint)
      - (i.e., the difference in the values assigned to Vi and Vj can't be equal to the difference between i and j.

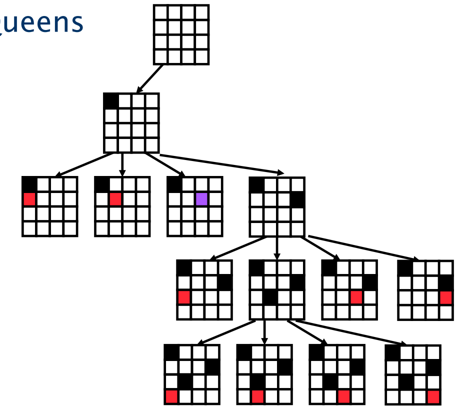
Example.

- 4X4 Queens



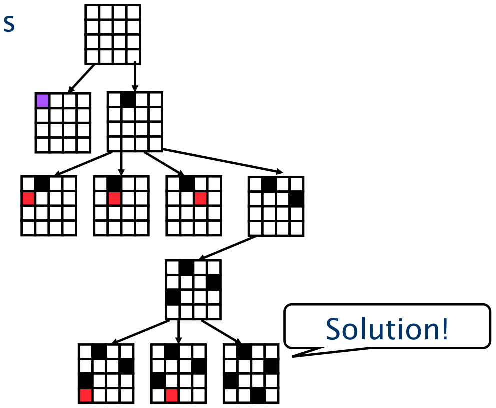
Example.

- 4X4 Queens



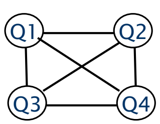
Example.

- 4X4 Queens



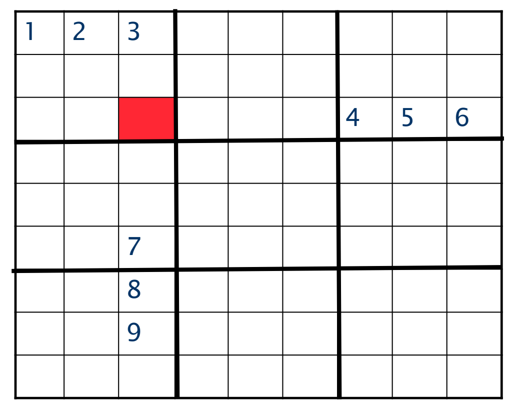
Backtracking Search

- Unary Constraints (over one variable)
  - e.g. C(X):X=2 C(Y): Y>5
- Binary Constraints (over two variables)
  - e.g. C(X,Y): X+Y<6
  - Can be represented by Constraint Graph
    - Nodes are variables, arcs are show constraints.
    - E.g. 4-Queens:



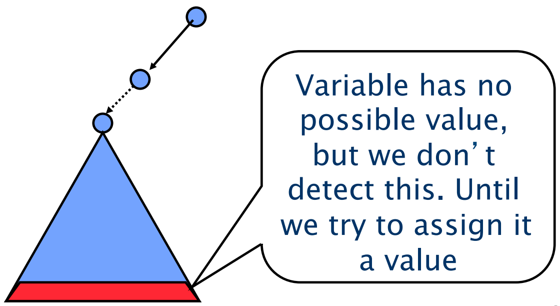
- Higher-order constraints: over 3 or more variables
  - We can convert any constraint into a set of binary constraints (may need some auxiliary variables)

Problems with plain backtracking.



Constraint Satisfaction Problems

- Sudoku:
  - The 3,3 cell has no possible value. But in the backtracking search we don't detect this until all variables of a row/column or sub-square constraint are assigned. So we have the following situation



Constraint Propagation

- Constraint propagation refers to the technique of "looking ahead" in the search at the as yet unassigned variables.
- Try to detect if any obvious failures have occurred.
- "Obvious" means things we can test/detect efficiently.
- Even if we don't detect an obvious failure we might be able to eliminate some possible part of the future search.

Constraint Propagation

- Propagation has to be applied during search. Potentially at every node of the search tree.
- If propagation is slow, this can slow the search down to the point where using propagation actually slows search down!
- There is always a tradeoff between searching fewer nodes in the search, and having a higher nodes/second processing rate.

Forward Checking

- Forward checking is an extension of backtracking search that employs a "modest" amount of propagation (lookahead).
- When a variable is instantiated we check all constraints that have only one uninstantiated variable remaining.
- For that uninstantiated variable, we check all of its values, pruning those values that violate the constraint.

Forward Checking

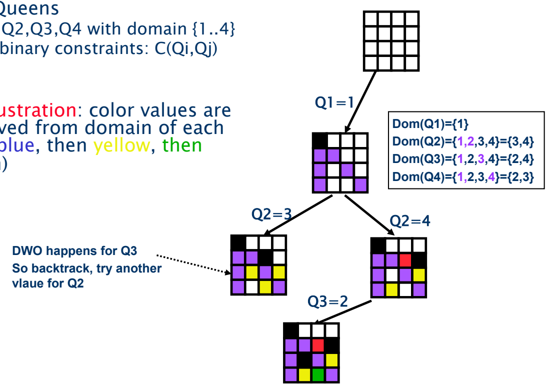
```
FCCheck(C,x)
  // C is a constraint with all
  // its variables already
  // assigned, except for variable x.
  for d := each member of CurDom[x]
    if making x = d together with
      previous assignments to
      variables in scope C falsifies C
    then
    remove d from CurDom[V]
  if CurDom[V]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


Forward Checking

```
FC(Level) (Forward Checking)
   If all variables are assigned
      PRINT Value of each Variable
      RETURN or EXIT (RETURN for more solutions)
                     (EXIT for only one solution)
   V := PickAnUnassignedVariable()
   Variable[Level] := V
   Assigned[V] := TRUE
   for d := each member of CurDom(V)
      Value[V] := d
      for each constraint C over V that has one
          unassigned variable in its scope X.
          val := FCCheck(C,X)
          if(val != DWO)
            FC(Level+1)
      RestoreAllValuesPrunedByFCCheck()
   return;
```

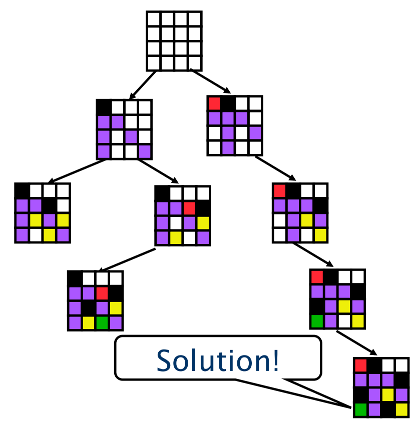

FC Example.

- 4X4 Queens
  - Q1,Q2,Q3,Q4 with domain {1..4}
  - All binary constraints: C(Qi,Qj)
- FC illustration: color values are removed from domain of each row (blue, then yellow, then green)



Example.

- 4X4 Queens continue…



Restoring Values

- After we backtrack from the current assignment (in the for loop) we must restore the values that were pruned as a result of that assignment.
- Some bookkeeping needs to be done, as we must remember which values were pruned by which assignment (FCCheck is called at every recursive invocation of FC).

Minimum Remaining Values

- FC also gives us for free a very powerful heuristic
  - Always branch on a variable with the smallest remaining values (smallest CurDom).
  - If a variable has only one value left, that value is forced, so we should propagate its consequences immediately.
  - This heuristic tends to produce skinny trees at the top. This means that more variables can be instantiated with fewer nodes searched, and thus more constraint propagation/DWO failures occur with less work.

Empirically

- FC often is about 100 times faster than BT
- FC with MRV (minimum remaining values) often 10000 times faster.
- But on some problems the speed up can be much greater
  - Converts problems that are not solvable to problems that are solvable.

Arc Consistency (2-consistency)

- Another form of propagation is to make each arc consistent.
- C(X,Y) is consistent iff for every value of X there is some value of of Y that satisfies C.
- Can remove values from the domain of variables:
  - E.G. C(X,Y): X>Y Dom(X)={1,5,11} Dom(Y)={3,8,15}
  - For X=1 there is no value of Y s.t. 1>Y => remove 1 from domain X
  - For Y=15 there is no value of X s.t. X>15, so remove 15 from domain Y
  - We obtain Dom(X)={5,11} and Dom(Y)={3,8}.
- Removing a value from a domain may trigger further inconsistency, so we have to repeat the procedure until everything is consistent.
  - For efficient implementation, we keep track of inconsistent arcs by putting them in a Queue (See AC3 algorithm in the book).
- This is stronger than forward checking. Why?

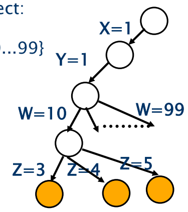
Backjumping

- Standard backtracking backtracks to the most recent variable (1 level up).

- Trying different values for this variable may have no effect:
  - E.g. C(X,Y,Z): X ≠Y & Z>3   and C(W): W mod 2 =0
  - Dom(X)=Dom(Y)={1..5}, Dom(Z)={3,4,5} Dom(W)={10…99}

  After assigning X=1,Y=1, and W=10,
  every value of Z fails. So we backtrack to W.
  But trying different values of W is useless,
  X and Y are sources of failure!

  We should backtrack to Y!



- More intelligent: Simple Backjumping backtracks to the last variable among the set of variables that caused the failure, called the conflict set. Conflict set of variable V is the set of previously assigned variables that share a constraint with V. Can be shown that FC is stronger than simple backjumping.

- Even a more efficient approach: Confilct-Directed-Backjumping: a more complex notion of conflict set is used: When we backjump to Y from Z, we update the conflict set of Y: conf(Y)=conf(Y) U Conf(Z)-{Z}