CSC384: Intro to Artificial Intelligence
Backtracking Search II

● Announcements

■ A1 marking scheme will be posted on the web by tomorrow.
■ Please let me know of any error in your marks.
■ A2 has been posted. Due Nov 1. Please start early to get help you might need
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:

  
  ![Diagram](image)

  Variable has no possible value, but we don’t detect this. Until we try to assign it a value.
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

$\text{FCCheck}\left( C, x \right)$

// $C$ is a constraint with all
// its variables already
// assigned, except for variable $x$.

for $d :=$ each member of $\text{CurDom}[x]$
    if making $x = d$ together with
    previous assignments to
    variables in scope $C$ falsifies $C$
    then
        remove $d$ from $\text{CurDom}[V]$
        if $\text{CurDom}[V] = \{}$ 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)

DWO happens for Q3
So backtrack, try another value for Q2
Example.

- **4X4 Queens continue...**

```
Solution!
```
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 (minimal 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 $Y$ 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. \( C(X,Y,Z): X \neq Y \) & \( Z > 3 \) and \( C(W): W \mod 2 = 0 \)
  - \( \text{Dom}(X) = \text{Dom}(Y) = \{1..5\}, \text{Dom}(Z) = \{3,4,5\} \) \( \text{Dom}(W) = \{10...99\} \)

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: **Conflict-Directed-Backjumping**: a more complex notion of conflict set is used: When we backjump to \( Y \) from \( Z, \) we update the conflict set of \( Y: \text{conf}(Y) = \text{conf}(Y) \cup \text{Conf}(Z) - \{Z\} \)