1 Backtracking Search

The pseudocode for backtracking search (BT) follows. Note that backtracking search is simply depth-first search with backtracking on the search tree of partial assignments as its nodes. The input is a (partial) assignment $A$ and a list of unassigned variables $U$. The output is either failure or a (complete) assignment. We denote the domain of variable $X$ as $D(X)$. The algorithm is initially called with $A$ the empty assignment and $U$ the list of all the variables of the CSP.

```plaintext
BT(A, U)
if A is complete then
    return A
end if
Remove a variable $X$ from $U$
for all values $x \in D(X)$ do
    if $X = x$ is consistent with $A$ according to the constraints then
        Add $X = x$ to $A$
        result ← BT(A, U)
        if result ≠ failure then
            return result
        end if
        Remove $X = x$ from $A$
    end if
end for
return failure
```

1.1 Forward Checking

The pseudocode for backtracking search with forward checking (BT+FC) follows. The algorithm uses an extra input $D$ that corresponds to the current domains of the variables. Initially, $D$ corresponds to the set of original domains.
BT+FC(A, U, D)
if A is complete then
    return A
end if
Remove a variable X from U
for all values x ∈ D(X) do
    if X = x is consistent with A according to the constraints then
        Add X = x to A
        D′ ← D (Save the current domains)
        for all Y ∈ U (i.e., Y an unassigned variable), Y → X (i.e., Y a neighbor of X in the constrained graph) do
            Remove values for Y from D′(Y) that are inconsistent with A
        end for
        if for all Y ∈ U, Y → X, we have D′(Y) not empty then
            result ← BT+FC(A, U, D′)
            if result ≠ failure then
                return result
            end if
        end if
    Remove X = x from A
end if
end for
return failure

2 Constraint Propagation

The pseudocode for the arc consistency algorithm (AC), the most basic form of constraint propagation, follows. The basic idea of the algorithm is to just check that all the arcs in the constraint graph are consistent. If not, it eliminates values from the domain of the variable that make the arc inconsistent, and insert to the queue arcs that are affected by that change in domain values. We denote that there is an arc from variable X to variable Y in the constraint graph by X → Y.

AC
Add all arcs in the constraint graph to Q
while Q is not empty do
    Remove first arc X → Y from Q
    if REMOVE-INCONSISTENT-VALUES(X, Y) then
        for all variables Z ≠ Y such that Z → X do
            Add Z → X to Q (if not there already)
        end for
    end if
end while
**Remove-Inconsistent-Values**\((X, Y)\)

for all values \(x \in D(X)\) do

    if there does not exist a value \(y \in D(Y)\) such that \(X = x, Y = y\) is consistent with the constraint between \(X\) and \(Y\) then

        Delete \(x\) from \(D(X)\)

    end if

end for

if deleted values then

    return yes

else

    return no

end if

The running time of the algorithm is \(O(ed^3)\) where \(e\) is the number of constraints and \(d\) is the number of values for each variable (i.e., the size of the domain). The running time analysis is as follows. Every arc is inserted into the queue \(Q\) at most \(d\) times, since \(d\) is the number of values for each variable and the algorithm eliminates at least one value from the domain of some variable at each round (otherwise, if no value is removed for the arcs in the queue \(Q\), then \(Q\) becomes empty and the algorithm stops). Therefore, the maximum number of iterations before \(Q\) becomes empty is \(O(ed)\). Checking that an arc is consistent at each iteration takes \(O(d^2)\). Therefore, the total running time \(O((ed)(d^2))\).

You should contrast this worst-case running time, which is linear in \(n\), to that of backtracking search which is \(O(d^n)\), which is exponential in \(n\). Hence, we can apply constraint propagation to try to reduce the size of the domains before we do backtracking since it does not take that long to run. Alternatively, you can run constraint propagation at every partial assignment node of the search tree. Usually in this case, however, constraint propagation is run only when there is only one value left for the variable just extended.